HUMPHRIES

ISSN 0265-2919

90p 78

# THE HOME COMPUTER ADVANCED COURSE

## MAKING THE MOST OF YOUR MICRO

An ORBIS Publication

UK £1.10 Aus $2.15 NZ $2.65 SA R2.45 Sing $4.90

# CONTENTS


APPLICATION

**THE MEANS OF PRODUCTION** We wrap up our series on micros in industry with a look at how computer controlled machines produce the finished product 1541

HARDWARE

**THE GREAT CONTENDER** Atari's 520ST, due for general release in September, looks likely to break open the serious home user market 1549

SOFTWARE

**LEARNING YOUR LINES** We delve into the use of condition testing as part of our interactive character programming 1544

COMPUTER SCIENCE

**ARRAY OF INTELLIGENCE** How FORTH can be extended to deal with array structures 1555

JARGON

**FROM S-100 BUS TO SDLC** A weekly glossary of computing terms 1548

PROGRAMMING PROJECTS

**BOWING OUT** Suggestions for refining our completed Go game 1546

MACHINE CODE

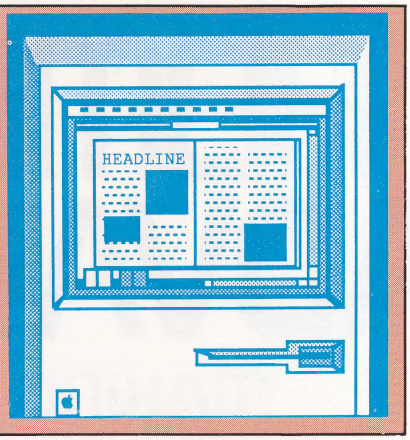
**SAFE KEEPING** A look at those routines used by the Amstrad operating system that control file transfer 1558

WORKSHOP

**POINT TO POINT** We begin constructing our digital multimeter by laying out the passive components 1552


## Next Week

- **We begin a new programming project series in which we will be designing a spreadsheet.**
- **Following on from our computers in industry series, we discuss one area in which computers look like having an enormous impact — the publishing business.**
- **The IBM PC/AT is a high speed version of the extremely successful PC that could be as successful as its predecessor.**




## QUIZ

1) GEM is the 'front end' of the Atari 520ST operating system. What is the name of the operating system and what other product does it resemble?

2) What is an 'actuator' and where would we use it?

3) Describe the difference between character and block transfers.

**Answers To Last Week's Quiz**

**1)** The Bitstik system requires that the BBC Micro be equipped with twin 80-track disk drives and a 6502 second processor.

**2)** The eight Kbytes of information from the Macintosh are used by the LaserWriter to create a bit-mapped image within the 1.5 Megabytes of memory.

**3)** The effect of POKE &DDDD, PEEK(&DDDD) is to copy a byte from a ROM into user memory. The effect of copying all the ROM into memory is that the user can 'customise' programs normally held in firmware.

## Coming Up

- A series that will take an in-depth look at the hardware of a range of computers, from minis to mainframes.
- A guide to the 68000 assembly language, which is used on the Sinclair QL.


**NON-VIOLENT ACTION?** More evidence to prove that you can't judge a game by its inlay INSIDE BACK COVER



**Editor** Stephen Cooke; **Art Editor** Claudia Zeff; **Deputy Editor** Steve Colwill; **Production Editor** Bobby Pickering; **Designer** Julian Dorr; **Staff Writer** Steve Malone; **Art Assistant** Caroline Clayton; **Sub Editors** Jon Kaye, Nik Lumsden; **Contributors** Chris Honey, Chris Laing, Bobby Pickering, Steve Cooke, Helena Sturridge, Marcus Jeffery, Steve Colwill, Steve Malone, Steven Vickers, David Mudd; **Software Consultants** Pilot Software City; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Maurice Geller; **Production Assistant** Susan Brown; **Subscription Manager** Christine Allen; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 14 Rathbone Place, London W1P 1DE; **© APSIF Copenhagen 1985; © Orbis Publishing Ltd 1985: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Heanor Gate Printing Ltd, Derby**

**HOW TO OBTAIN ISSUES AND BINDERS FOR THE HOME COMPUTER ADVANCED COURSE** – Issues can be obtained by placing an order with your newsagent or direct from our subscription department. If you have any difficulty obtaining any back issues from your newsagent, please write to us stating the issue(s) required and enclosing a cheque for the cover price of the issue(s). **AUSTRALIA** – please write to: Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G, Melbourne, Victoria 3001. **MALTA, NEW ZEALAND & SOUTH AFRICA** – Back numbers are available at cover price from your newsagent. In case of difficulty, write to the address given for binders.
**UK/EIRE** – Issue Price: 90p/IR£1.15. Subscription: 6 months: £26.00. 1 Year: £52.00. Binder: please send £3.95 per binder, or take advantage of our special offer in early issues. **EUROPE** – Issue Price: 90p. Subscription: 6 months air: £44.72. Surface: £36.14. 1 year air: £89.44. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **MALTA** – Obtain binders from your newsagent or Miller (Malta) Ltd, MA Vassalli Street, Valetta, Malta. Price: £3.95. **MIDDLE EAST** – Issue Price: 90p. Subscription: 6 months air: £50.18. Surface: £36.14. 1 year air: £100.36. Surface: £72.28. Binder: £5.00. Airmail: £8.25. **AMERICAS/ASIA/AFRICA** – Issue Price: US/CAN$1.95/90p. Subscription: 6 months air: £59.54. Surface: £36.14. 1 year air: £119.08. Surface: £72.28. Binder: £5.00. Airmail: £9.50. **SOUTH AFRICA** – Issue Price: SA R2.45. Obtain binders from any branch of Central News Agency or Intermag, PO Box 57394, Springfield 2137. **SINGAPORE** – Issue Price: Sing $4.50. Obtain binders from MPH Distributors, 601 Sims Drive, 03-07-21, Singapore 1438. **AUSTRALASIA/FAR EAST** – Issue Price: 90p. Subscription: 6 months air: £64.22. Surface: £36.14. 1 year air £128.44. Surface: £72.28. Binder: £5.00. Airmail: £9.75. **AUSTRALIA** – Issue Price: Aus$2.15. Obtain binders from First Post Pty Ltd, 23 Chandos Street, St Leonards, NSW 2065. **NEW ZEALAND** – Issue Price: NZ$2.65. Obtain binders from your newsagent or Gordon & Gotch (NZ) Ltd. PO Box 1595, Wellington.
**ADDRESS FOR BINDERS AND BACK ISSUES** – Orbis Publishing Limited, Orbis House, Bedfordbury, London WC2 4BT. Telephone 01-379 5211. Cheques/postal orders should be made payable to Orbis Publishing Limited. Binder prices include postage and packing and prices are in stirling. Back issues are sold at the cover price, and we do not charge carriage in the UK.
**NOTE** – Binders and back issues are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK and Australian markets only. Binders and Issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.
**ADDRESS FOR SUBSCRIPTIONS** – Orbis Publishing Limited, Hurst Farm, Baydon Road, Lambourn Woodlands, Newbury Berks, RG16 7TW. Telephone: 0488-72666. All cheques/postal orders should be made payable to Orbis Publishing Limited, Postage and packaging is included in subscription rates, and prices are given in sterling.

COVER PHOTOGRAPHY BY CHRIS STEVENS ATARI 520ST COURTESY OF THE SILICA SHOP: TEL 01-309 1111

# THE MEANS OF PRODUCTION

**Today's manufacturing methods depend on translating the precise drawings produced with the aid of CAD, into the motions of the machines that will assemble the finished products. In this instalment, we look at two techniques used to achieve this.**

In the past, engineers read the drawings and set their machine tools by hand. Numerical control (NC) machines introduced a faster and more accurate process: the machine tool would set itself in accordance with instructions received via a command language. The designers would produce technical drawings incorporating all the appropriate measurements. Production engineers then translated the drawings into NC programs. The machine tool would then have all the necessary information to cut the part.

NC programmers have an industry standard set of commands to define the types of movement a machine can do. The programmer picks a movement and then adds the precise co-ordinates. He will also need to define the speed at which the machine is spinning or turning. Complex jobs are often done in stages, proceeding through a squared-off shape to the finished part. This system is now gradually being superseded by computer numerically controlled machines (CNC) in which the paper tape which has always fed NC machines is replaced by magnetic tape or floppy disks. In the most sophisticated of these systems, companies are looking at the possibility of moving directly from CAD to automatically-generated NC programs.

Not all CAD systems are sufficiently accurate to do this, but where they are, full use can be made of the central CAD database at all stages of design and production. The central CAD database, with full geometric description of the part, can be accessed by the production engineer, who has a library of machine tools at his disposal. He chooses the one needed to cut the part and, as he writes the program, he can see the tool moving along the contours of the drawing. In this way simple but expensive errors — such as directing the tool forwards instead of backwards — can be avoided.

The use of the same database also speeds up the process, since any changes made by the designer during the course of development are instantly relayed to the engineer. Failures in the design which might make it impossible to manufacture a part can be referred straight back to the designer.

Modern process control is essential to the smooth functioning of nearly all industry, and is usually carried out by means of scores of microprocessors. A process control system is made up of five crucial components:

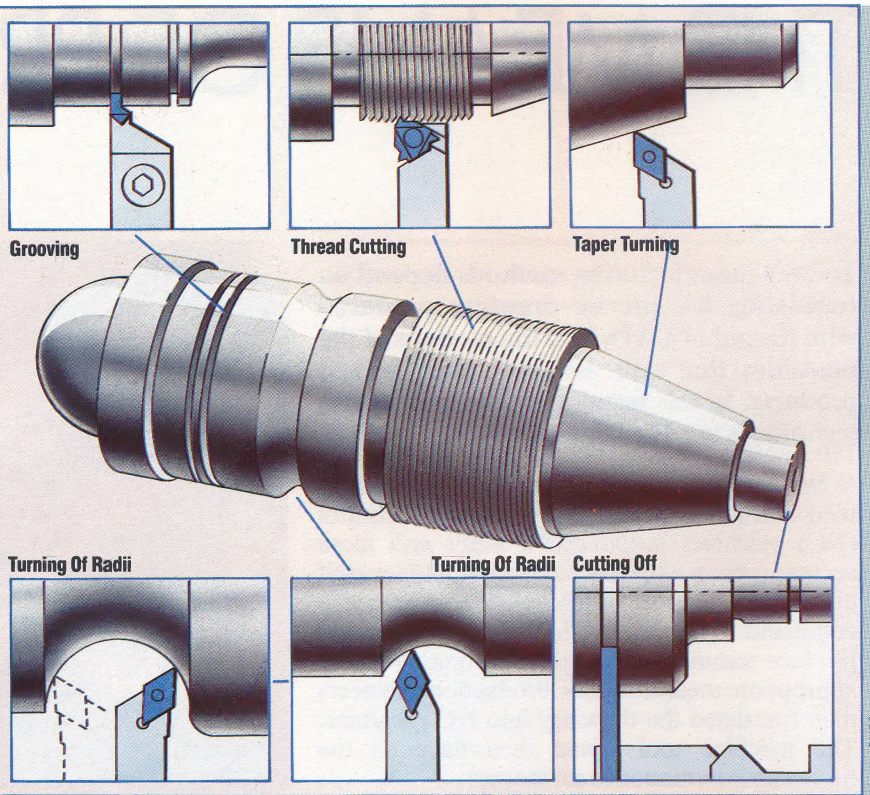
- A *sensor* to measure the particular characteristic, such as temperature, fluid flow or pressure.
- An *actuator* that can initiate an action in response to a signal.
- There must also be a *control valve*, or its equivalent. This will be driven by the actuator to effect the process.
- A *controller* determines what action should be taken, based on information received from the sensor.

COURTESY OF DR. JOHANNES HEIDENHAIN, W. GERMANY

**On The Job**
Advances in natural language processing have enabled programmers to construct front-end user interfaces for CNC machines that will accept job specifications in plain English. Earlier NC machines were not so efficient, needing a complex series of codes to be entered by highly trained personnel. In future, blending CNC and robot technology could produce 'intelligent' machines capable of performing a variety of different tasks in different locations within the factory

## Precision Tools

CNC machines can perform a variety of different operations upon the same component by selecting different cutting tools as dictated by the master program, entered either by the engineer or directly from the central computer system. Microprocessor control enables a single machine to master all the major cutting techniques of a conventional lathe with minimal supervision




KEVIN JONES

## Process Control

The real work of a process control system is done by the process sensors. These are the many different devices which read, measure and interpret data. Most of them contain microprocessors.

**Temperature Measurement**

- Thermocouple: The cheapest and most common measure of temperatures. A thermocouple uses two different metals with different rates of expansion under heat. Between them they produce a voltage, which is the basis of an analogue signal.
- Resistance Thermometer: This is based on the principle that heat increases the electrical resistance in conducting wires. The current is passed through one heated and one unheated wire. The difference in voltage is measured and the change in resistance calculated.
- Radiation Pyrometer: Measures radiation heat and is used for higher temperatures than the previous two devices. The pyrometer focuses radiated heat on to what are essentially thermocouple devices and measures the temperature from there.

**Flow Measurement**

- Differential Pressure Meters: These calculate the rate of flow by measuring the pressure at different points on either side of an obstruction.

**Pressure Measurement**

- Bourdon Tube: This uses a metal tube shaped into a C, a spiral or a helix. When pressure is applied to one end, the tube tries to straighten out. The pressure is measured from how far the free end moves.
- Manometer: A U-shaped tube with one wider arm. A float is mounted in the wider arm. Pressure in the longer, thinner arm changes the surface level in the other.
- Diaphragm: A mechanical linkage measures the movement exerted on a thin rubber diaphragm. A more complicated sensor is in the bellows, where it is transferred to the bottom and measured.

**Level Measurement**

- Float-Operated Devices: Work on a principle

- To do this the *transmitter* must take the signal from the sensor and pass it to the controller in a form it can 'understand'.

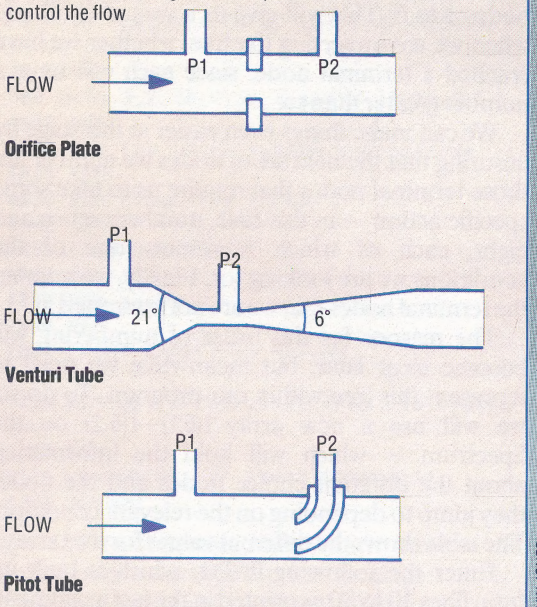
Most of these are analogue devices, carrying their information in the form of voltages. For this reason the system contains an analogue-to-digital converter which changes the signal to a form acceptable to the computer. Process control systems depend on such devices sitting at key points in the process plant and are nearly all concerned with running the same program over and over again. It might, for instance, have to run a program every second to check the rate of flow of a fluid through a pipe. If the rate exceeds the limits set, then control valves have to be activated.

Although process control systems are as varied as the processes they control, they can be divided into two main types: continuous control and sequence control. These are not mutually exclusive and may, indeed, be included in the same system.

Continuous control sees raw materials fed into a process and passed through without stopping.

## Quick Flow

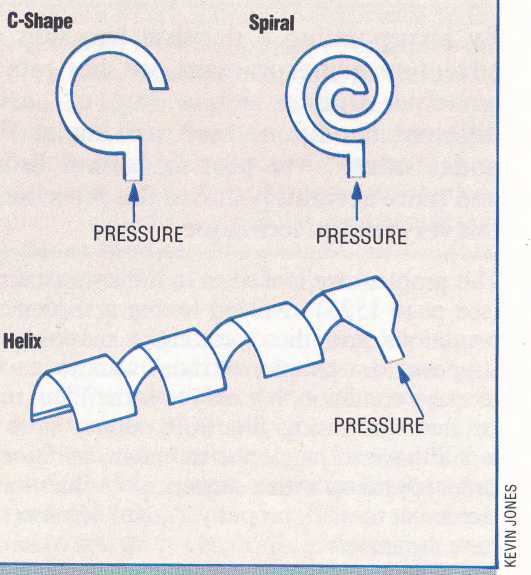
Flow of liquids and gasses can be measured and transmitted to a host computer by monitoring differential pressure. Pressure is measured in each case at two points, P1 and P2. The differential pressure is equal to P1-P2, and the flow in the tube is directly proportional to this figure. A computer can therefore monitor and control the flow




KEVIN JONES

## Under Pressure

The Bourdon tube consists of a metal tube which, when pressure is applied to the device, tends to straighten. Movement of the tube is then measured to give an indication of pressure and the result can be transmitted to the computer for analysis. Bourdon tubes come in three forms — C-shape, spiral and helix




KEVIN JONES

rather like the fishing float. Level changes are measured as the float moves up and down.

- Pressure Devices: The pressure at the bottom of a tank is of course a measure of the amount — and, therefore, the level — of liquid in the tank.
- Radiation: A cell on one side of the vessel calculates how much gamma radiation is getting across from a source on the other side. Radiation will only get across where the liquid does not bar its way and so the level can be measured.
- Ultrasonic Sensor: This measures depth by timing the response from an ultrasonic signal.

### Weight Measurement

- Electrical Resistance Strain Gauges: These use the fact that electrical conductors change their resistance when they are strained.
- Semiconductor Gauges: Semiconductor materials, such as silicon, produce a voltage under pressure and can therefore be used as a measure.

### Chemical Analysis

- Infra-red: This form of analysis is used for measuring gases. Samples of the gas are passed through an infra-red source to an infra-red detector. The amount of infra-red blocked on the way across is a measure of the amount of gas present.
- Thermal Conductivity Analysers: Changes in a gas alter its ability to conduct heat. The gas is passed through a heating element at a constant rate; if the conduction of heat changes, it is possible to infer what has happened to the gas.

### Speed

- Tachogenerator: This measures a spinning speed. A tacho is attached to a rotating shaft and as this spins it generates current. Since the current is proportional to speed it is possible to measure the latter.

Output from these and many other sensors in process control systems is fed into a three-term controller. This takes the measured value from the sensors and compares it to a value set by the operator. That is the level at which it is supposed to rake action. If the value is above or below the set point, it can activate the valve and change the process in any way required

Control is needed of the rate of flow along the way. The system must be able to respond to small changes in order to maintain optimum conditions. With sequence control, however, the process is controlled in batches. The raw material has to go through a number of physical or chemical changes, so control concerns itself with initiating a series of actions at the right time and under the right conditions. It must also control all the different batches to ensure that each gets the same treatment.

The great advantage of using computers in process control is that they make it possible to do many things at the same time. The particular processes can be individually controlled by standard components which can be swapped around in the event of a break down. Above all, computerised process control allows just one plant supervisor to control many different operations from a single terminal. Instead of having to read hundreds of dials and meters he can keep track of what is happening by looking at the 'mimic diagrams'— stylised representations of the plant and its processes — displayed on his VDU.

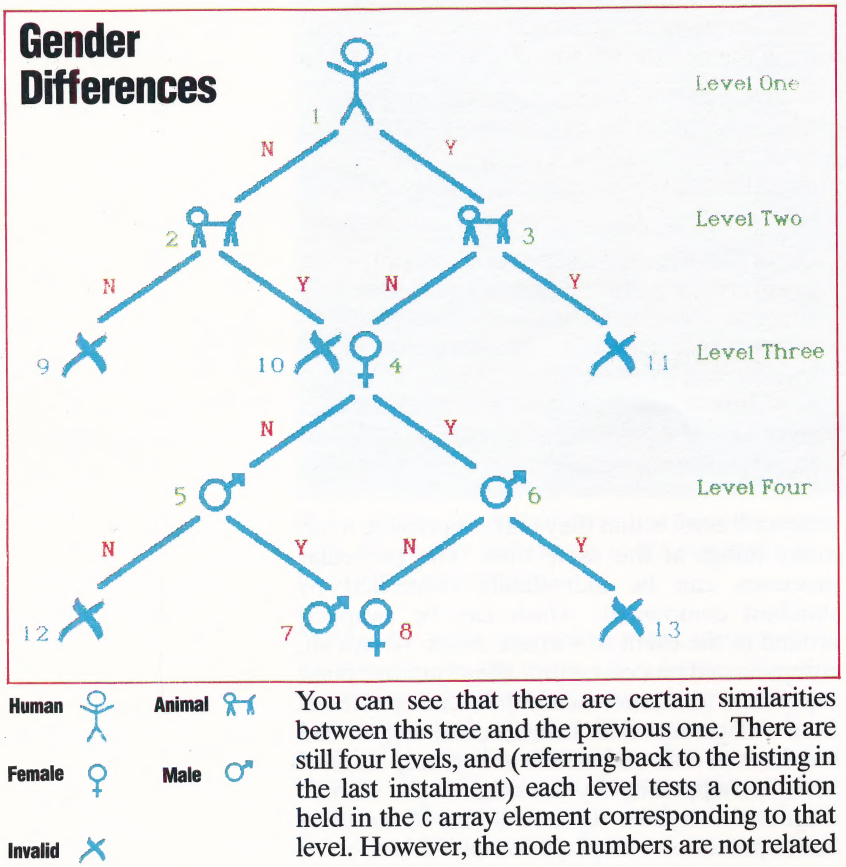
# LEARNING YOUR LINES

**By incorporating a decision tree into our adventure game program, we can provide ourselves with a simple way of sorting different conditions and arriving at final node 'values'. We provide several listings and more accurately define the processes of this very fruitful technique.**

The problem we looked at in the last instalment (see page 1526) involved testing a sequence of conditions and then printing a message. But suppose, for example, we didn't want to respond to every condition, but only to some? This might be awkward using ON...GOTO alone, since we would have to juggle the different conditions in order to end up with a sequence of values for the statement to work properly. Again, decision trees have the answer.

Going back to our previous example of four conditions — indicating human, animal, male or female — the following diagram shows a tree designed to check the different conditions and reject all entries by the user, except those confirming that the user is either a male or a female human being.

**Men And Women Only**
Descending through the tree shown here will bring the user to an 'invalid' terminal node unless the questions have been answered by a male or female human. Although the design is similar to the example in our previous instalment (see page 1527), this tree has terminal nodes at different levels – not just at level 5. Choice nodes have been numbered in blue, terminal nodes in red and black. The red numbers indicate valid results (i.e. male or female humans)




You can see that there are certain similarities between this tree and the previous one. There are still four levels, and (referring back to the listing in the last instalment) each level tests a condition held in the c array element corresponding to that level. However, the node numbers are not related as before. This is because a terminal node may not necessarily be at the very bottom of the tree, but may occur on one of the earlier levels — as is the case with node 9, for example.

To integrate this new tree into our program, we must first number the choice nodes; in this case, nodes 1 to 6. This will give us a way of checking, when we are traversing the tree, whether we have reached a terminal node, since each will have a number greater than six.

We can make things even easier at this stage by ensuring that the next set of nodes we number are those terminal nodes that require us to take some specific action — in this case, numbers seven and eight, each of which represents one of the conditions we are looking for. Finally, we number the terminal nodes that we are not interested in.

The reasons for this order of numbering will become clear later, but meanwhile we need to represent this tree within our program. To do so, we will use a new array t(6,1)—t(6,2) on the Spectrum — which will hold the information about the different choice nodes and the nodes they jump to depending on the relevant condition. The table shows the different values for the t array.

Enter the following listing, which is built up from lines 10 to 80 as printed in the last instalment, and the low-level routines from our Dog and Bucket program. Spectrum, Commodore 64 and BBC Micro flavours for the low-level routines were printed on page 1506. Note that we have squeezed in some new lines between lines 20 and 30, and added a line number 500. These new lines read the relevant data into the array t. We have also added a DIMension statement onto the end of line 10, DIM t(6,1).

## Tree Table

The array t(6,1) — t(6,2) on the Spectrum (which does not allow zero elements in arrays) — holds the necessary data for our tree structure. If, on reaching node 4, for example, we find that the condition held in c(4) is false, we would branch to the node number held in the array member t(4,0). It it was true, the branch would be made to the number held in t(4,1). Spectrum owners would branch using t(4,1) and t(4,2) respectively

| Node Number | False | True |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 9 | 10 |
| 3 | 4 | 11 |
| 4 | 5 | 6 |
| 5 | 12 | 7 |
| 6 | 8 | 13 |

```
10 h$="human ":a$="animal ":m$="male ":f
$="female ":q$="Are you ": DIM c(4),t(6,
1)
20 GOSUB 4050: REM clear the screen
22 REM set up tree array
24 c=6: REM this is number of choice nod
es
26 FOR x=1 TO c: READ t(x,1),t(x,0): NEX
T x
```

```
30 REM set up the four variables
40 PRINT q$;h$;: INPUT i$: c(1)=ABS(i$="
y" OR i$="Y")
50 PRINT q$;a$;: INPUT i$: c(2)=ABS(i$="
y" OR i$="Y")
60 PRINT q$;f$;: INPUT i$: c(3)=ABS(i$="
y" OR i$="Y")
70 PRINT q$;m$;: INPUT i$: c(4)=ABS(i$="
y" OR i$="Y")
80 PRINT
500 DATA 3,2,10,9,11,4,6,5,7,12,13,8
4000 REM
4010 REM low level system subroutines
4020 REM
4030 REM clear the screen
4040 REM
4050 CLS: RETURN
4060 REM
4070 REM beep
4080 REM
4090 PRINT CHR$(7);: RETURN
4100 REM
4110 REM get a character from the keyboa
rd
4120 REM
4130 i$=INKEY$: IF i$="" GOTO 4130
4140 RETURN
```

Now add the following lines, which traverse the tree and print out an appropriate message:

```
90 GOSUB 210: REM call tree sort routine
100 PRINT "Press a key to continue..."
110 GOSUB 4130: REM get a character
120 GOTO 40
200 REM sort tree
210 n=1: k=1: REM start at node 1, level
 1
220 n=t(n,c(k)):k=k+1: IF n<=c THEN 220
230 ON n-c GOTO 250,260
240 PRINT "Invalid input...Please try ag
ain": RETURN
250 PRINT "You're a man...": RETURN
260 PRINT "You're a woman": RETURN
```

If you now RUN this program, you will find that it rejects all inputs except those made by 'truthful' male and female humans. The tree is traversed in a similar way to our first example, but with the following changes. First, we don't necessarily know how many levels we will descend through before encountering a terminal node, so we cannot use a simple FOR n TO number of levels loop. Instead, we initialise a new variable, k, in line 140 to keep track of the level we are on.

Having set up n to hold the start node number (1), we then descend through the tree in line 150, using the formula n=t(n,c(k)) to pick the next node number out of the array t. We then move down a level (k=k+1) and, if the current node number is less than or equal to the number of choice nodes (n=c), we know that we have not yet arrived at a terminal node, so we jump back to the beginning of the line and repeat the process.

You can now see why the nodes are numbered as described earlier. Dividing the terminal nodes into two categories — those that represented the two final conditions (male and female human) that we wanted to trap followed by those which we wanted to reject — enables us to use an ON...GOTO statement to print the different messages. As we've already pointed out, the main drawback of using ON...GOTO is that it requires a sequence of values that is often awkward to arrange, but the tree enables us to do it without difficulty.

So far we've concentrated on structures that have one thing in common — all the conditions tested on each level of a tree have been the same. This has simplified matters because we haven't had to find out what condition to test at each node. If, for example, we found ourselves at a node on level 3, we knew that the condition to be tested was held in c(3). Unfortunately, things aren't always that simple. As the number of conditions to be tested increases, the structure of the tree needed to deal with them inevitably becomes more complex. However — and this is one of the great benefits of using this method of testing conditions — more complex trees do not necessarily take up more space in your program.

Character-handlers can be as complex as you care to make them, and the best approach is to split up the problem into different modules, just as we have done with the main body of the program. The first area of character behaviour we'll look at is the way in which objects may be manipulated in the Dog and Bucket. At the very least, we need routines to get, drop, eat, drink, give, receive, and throw objects, and, of course, we need to tell the player (if present) what's going on.

The 'drop/throw/give' routines, for example, could be considered as a group, because they all depend on the character starting with an object and ending up without one. To determine which routine is the most appropriate, we obviously need to test such conditions as 'Is there someone in the room', and if so, 'Is that someone able/willing to receive the object'. We would also need to test a character's mood, to see whether or not objects are likely to be thrown and so on. Other complications include the need to introduce a random element into the process, and the need for other, associated routines — lowering a character's energy if, for example, one of them has been hit by a glass of beer! In the next instalment, we will design a tree that will take all these possibilities into account.

## Basic Flavours

**Spectrum:**

Flavours for lines 4000 to 4140 have already been given on page 1508. For this instalment, Spectrum owners should make the following additional changes:

```
10 h$="human ":a$="animal ":m$=" male ":
   f$="female ":q$="Are you ": DIM c(4) ,t(6, 2)
40 PRINT q$;h$;: INPUT i$: c(1)=ABS(i$=" y" OR
   i$="Y")+1
50 PRINT q$;a$;: INPUT i$: c(2)=ABS(i$=" y" OR
   i$="Y")+1
60 PRINT q$;f$;: INPUT i$: c(3)=ABS(i$=" y" OR
   i$="Y")+1
70 PRINT q$;m$;: INPUT i$: c(4)=ABS(i$=" y" OR
   i$="Y")+1
```

# BOWING OUT

**We complete our Go programming project by considering a number of additions that can be made to the listing. We look at how a current game can be saved to disk or tape and subsequently reloaded, and how the computer's response time can be improved.**

There are many ways in which the programs presented could be improved. We'll consider a few of these before looking at methods used in Go programs on large systems, some of which could probably be implemented on microcomputers.

Spectrum, Commodore 64 and Amstrad owners will probably have noticed that these programs are translations from the original BBC Micro version. Though in many cases they have already been significantly changed to allow for differences in commands and program structure, they have purposely been kept as similar as possible. Rewriting these, using short variable names and taking more advantage of the features available on the specific machines, could signficantly improve their performance.

It is unlikely that the Spectrum and Commodore 64 versions will ever work as quickly as the BBC Micro due to their slower BASICs. However, you may like to consider converting some routines to machine code to speed up execution. The frequency with which routines are executed can easily be checked by placing a count variable at the start of each, then printing these out at the end of the game. Doing this you'll find that one of the most critical routines is PROCsearch, which is not only used frequently during the initial PROCgroup evaluation, but is also called at least once by each call to FNlegality. Having the board (board%) set up as a sequence of bytes, the conversion to machine code should be quite easy.

There are many other basic tactics in the game of Go that would certainly improve the program's play if they were included. These include such features as ladders, Joseki (opening play) and so on. One very important feature that could be implemented is that of 'life and death'. The program already incorporates an 'unconditional death' evaluation. This merely checks whether a particular stone group has no remaining liberties, removing it from the board accordingly. Taking this one stage further, you could easily assess whether a group has 'unconditional life'. To implement this we would have to:

1. Temporarily fill as many liberties of the group as legally possible with stones of the opposite colour.
2. Then use PROCsearch to count the remaining liberties of the group. If this value (clib%) is two or greater, then the group cannot be captured — unless, of course, the defender is stupid enough to fill his own liberties.

It should be noted that when filling liberties, they must all be tried to exhaustion. If they are tried in order, false eyes may not be filled. For instance, the group may have an internal liberty which can be filled only when all the external liberties are filled. Having filled one internal liberty, it may be possible to fill further false eyes and true eyes.

Having implemented unconditional life in a one group, you can move on to life of a group on the whole board. This allows for the possibility of groups sharing eyes. You'll find an example of this if you look back to the first part of the series (see page 1366). To implement this, you must first follow the procedure above. If an internal liberty is found which is surrounded on at least one side by a different friendly group, then the routine must call itself recursively to check the safety of this adjoining group and so on. Only if the adjoining groups are safe from capture is the first group's internal eye safe, thus affecting the safety of the original group.

Though the program could be significantly improved in many ways, it is unlikely that it will ever become a world-class player. Programs running on large systems still only play a little better than a novice, even after more than twenty-five years of research. There are a number of

## Well Saved

One useful additional feature would be to allow the game to be saved to tape or disk and resumed at a later date. This would involve writing a new PROCsave_game routine to be called after a player has typed QUIT. The save procedure would have to save all the game status variables and byte table values. These consist of:

| | |
|---|---|
| board% to board%+255 | Main board bytes |
| weight% to weight%+255 | The weighting table values may have been changed by PROCremove |
| move% | This not only gives the move number, but indicates which player is next — odd or even |
| capture%(2) | Number of stones captured by Black and White |
| ko% | Indicates the position of a ko point, or zero |

You might also optionally save:

| | |
|---|---|
| atari1$ and atari2$ | These indicate a customary 'atari' warning by Black or White |
| T$ | Method of Black's last move choice |

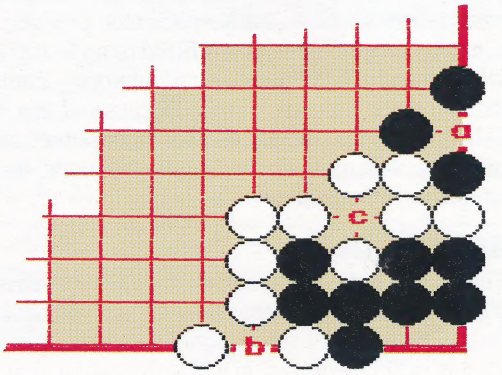
reasons for this.

The board itself causes a number of problems due to its size. A 19 by 19 board is usually said to have an order of magnitude of $3^{361}$, which is approximately $10^{172}$. This figure is calculated as the upper limit on the number of configurations of vacant black or white points on the board. A rough estimate of the average number of potential moves by a player is 250. If the traditional game-playing technique of tree searching were applied to Go, then an exhaustive search only three moves ahead would entail generating and evaluating about 8,000,000 board positions. Yet videotaped game records have shown that advanced amateur players can look a staggering 30 moves ahead, and Go literature frequently contains still deeper look-ahead sequences.

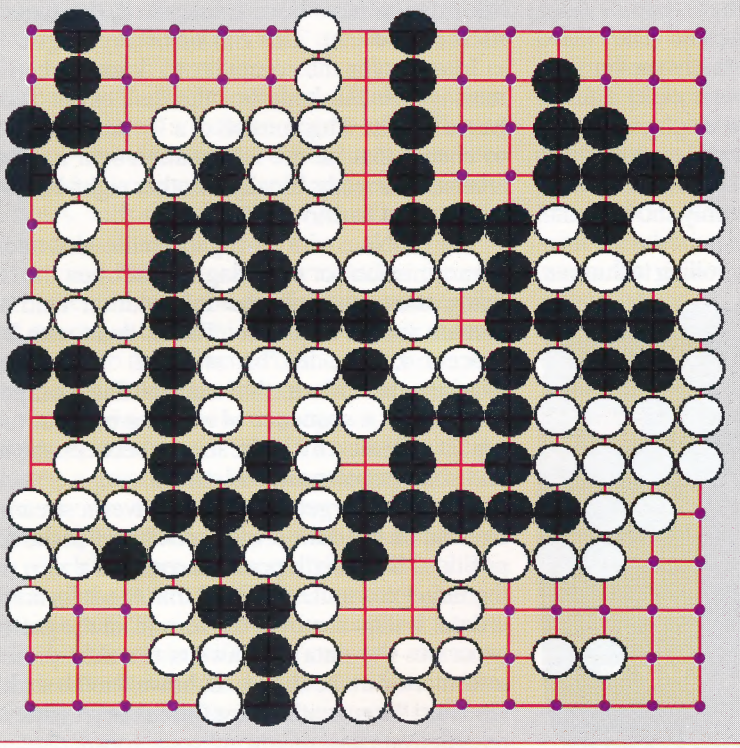
The solution to the problem of large trees is to use some form of 'focusing' function. This would initially perform a rough evaluation of the whole board, with the object of deciding which areas of the board are in critical situations. Having chosen one or two small battle areas, full-depth trees can be generated just for these stone patterns. The trees can also be severely 'pruned' using the alpha-beta algorithm, and by providing routines that can pick out critical points, eye positions and so on.

Another problem with Go is the imprecision of its rules. For instance, the end of the game is judged to be when neither player can gain any further advantage, but Go programs have great difficulty in determining this rather vague condition. A further problem concerns special ko conditions. For example, the diagram shows a 'triple ko' position. Black plays at 'b', capturing the white stone to the right. White than captures the black stone just below 'a', followed by Black capturing at 'c'. White can now play back into the position of the first capture, removing the black stone at ' a '. Black recaptures the white stone at 'a', and so on. During a normal game, if neither player is willing to give up the local fight by playing elsewhere, then the game is cancelled. Though unusual, even more complex ko situations can occur. In order to recognise them, a Go program will have to store all previous game positions so that they can be checked against the current position.

At present, Go programs do not play well. Nevertheless, not many years ago, few people reckoned that computers would ever play a good game of chess. The question now is whether or not a computer chess program will ever attain world-champion status. The fact that the Japanese have chosen Go as a suitable problem to test their Fifth Generation computers seems to indicate the era of the Go master is not too far distant.




## End Of Game Scoring



At the end of the game, the board must be evaluated and points allocated to each side according to the amount of territory captured.

The white group in the bottom-right corner of the board has surrounded 17 points of territory. The two white stones within this area are not included — only vacant points are counted.

The black group in the top-left corner is in fact two groups, as only a diagonal 'connection' exists between the two halves. However, all exits from the territory are cut off by black stones and so Black can add 17 to his score.

White surrounds the bottom-left corner but here a stranded black stone is present. The rules of Go state that this stone does not actually need to be captured during the game. It is at this point that the stone is removed, leaving 10 points of territory for White. An extra point is also added to White's score to account for the black stone thus removed.

Black appears to have captured the top-left corner but is itself entirely surrounded and cut off by a larger white group. The black group can therefore be removed from the board and White gains 18 territorial points and a further five points for the black stones captured. All other vacant points are neutral as they do not form areas of territory surrounded solely by one colour

CAROLINE CLAYTON

## S-100 BUS

Originally developed for the Altair computer (considered to be the first microcomputer), this bus system has been further developed for use by a wide range of micros. The *S-100 bus* is a 100-way connector which is used to connect a computer to its peripheral system or expansion cards. Although designed for use with eight-bit processors, it can now be used with a wide variety of processors, which is reflected in the variety of expansion boards performing different functions that have been developed for use with the bus. The boards are generally 125 by 250mm, although 250 by 250mm boards are now beginning to appear.

One of the features that has enabled the S-100 bus to be adapted to such a wide range of machines is that it uses a power supply direct from the computer's transformer. Power obtained in this way is said to be 'unregulated' (there may be fluctuations in the voltage). Although electronic components have their own built-in tolerance to power fluctuations, many transformers, particularly those on early machines, produce an irregular supply. Thus, cards fitted to an S-100 bus are required to have their own regulators — circuits that produce a steady supply. Because of this, the S-100 bus can be used with computers with different voltages, and all that will need to be altered is the 'regulator'.

## SCROLLING

This is when text or graphics are moved across a VDU screen either from side to side or up and down. As information appears on one side of the screen, other information will move out of view on the other side. The effect is rather like looking through a window that enables you to examine a small part of a large amount of information.

*Scrolling* is necessary when there is more video data than the screen can display. The video memory (which is what you actually see on your VDU screen), is usually between eight and 16 Kbytes. The data required to be displayed on the screen is held in the main memory. However, this is often larger than the capacity of the screen to hold it. Thus, the video memory can only show a small area or 'window' of memory.

There are two basic types of scrolling technique. In 'hard scrolling', a register will contain the address of the start of the window, which, as the screen is scrolled, is incremented or decremented in steps corresponding to the resolution of the screen. This has the effect of moving the area of main memory that refreshes the video RAM.

With 'soft scrolling', the technique used is somewhat different. In this system the window does not move. Thus, in order for the memory data to be displayed on the screen, the information has to be fetched from its location by the processor and transferred to the memory locations set aside for the screen window. In order to scroll, the processor simply moves further along the area of memory where the information is held, so that new data can be displayed on the screen.

**Moving Pictures**
Scrolling is a technique that has become widely used in arcade-style games, where the illusion of constant motion is produced on screen. The game shown here runs on the Commodore 64, a machine that uses a combination of hard scrolling and soft scrolling in its screen display. Here, the screen memory is a 'window', moving across the main memory, which produces the effect of scrolling

## SDLC

Known as 'synchronous data link control', *SDLC* is a protocol for data transfer between computers. The system is built around the concept of 'frames' which are used to tell the target computer when a message begins and ends.

The frame consists of an eight-bit flag which is 01111110 in binary. The protocol requires that the target computer knows a message is on the way when it receives a series of five ones consecutively (the sixth 1 in the flag byte is used as a check, which will be explained later). The next part of the frame to arrive is an eight-bit address which targets the message to one of 254 different computers that may be in the network. Address 0 is a dummy address for testing the system, while address 1 indicates a 'universal' message for all machines.

The third part of an SDLC frame is an eight-bit control field. These are used by computers within the network to transmit control information between each other. Once these three pieces of information have been transmitted, the message itself can be sent. This can be of any length.

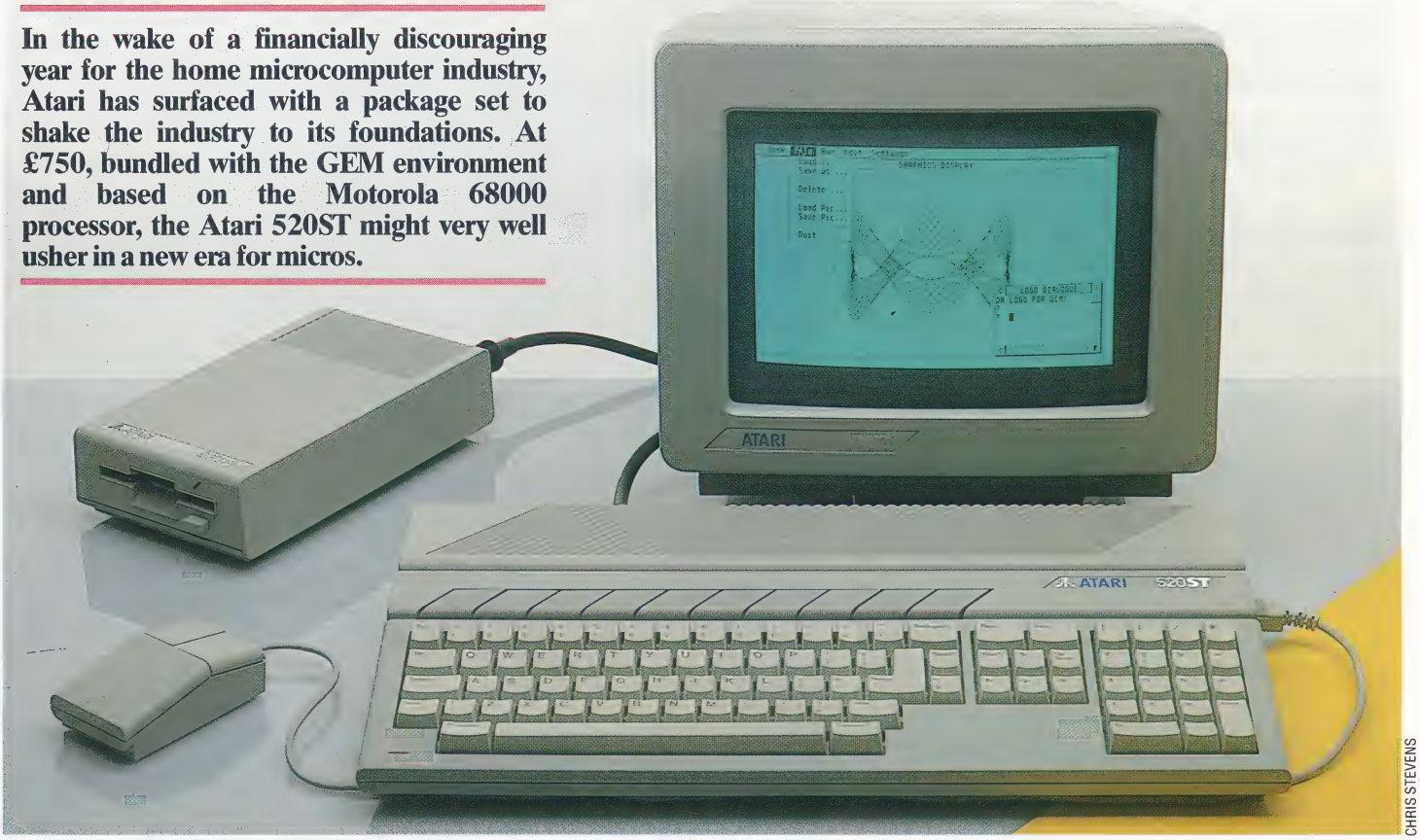
In order for the computer to 'know' when the message has finished, the other half of the 'frame' has to be sent. This consists of a 16-bit frame check to ensure that all the information sent has been received correctly. Finally, another eight-bit flag is transmitted to end the frame.

The problem with having six continuous ones as a transmission (or end) flag, is that often the data being transmitted in the information field will contain such a sequence. This is dealt with by a process known as 'bit stuffing'. The process requires that when the transmitting computer notices that a sequence of six ones is about to be sent, it will insert a 0 in the stream between the fifth and sixth positions in the stream.

When the target machine receives a stream of five one in the data, it will check for a 0 in the sixth position. If the sixth position contains a zero then the target computer 'knows' that the information is data. It then removes the zero from the pattern, processes the data and awaits the next piece of information in the stream. If, however, the sixth bit is a 1 and the seventh bit is a 0 then the bit pattern is regarded as a flag and the message is ended.

# THE GREAT CONTENDER

**In the wake of a financially discouraging year for the home microcomputer industry, Atari has surfaced with a package set to shake the industry to its foundations. At £750, bundled with the GEM environment and based on the Motorola 68000 processor, the Atari 520ST might very well usher in a new era for micros.**




CHRIS STEVENS

1985 is likely to go down as something of a watershed for the home microcomputer industry, the year when the almost exponential growth of previous years began to settle down and the industry began to mature. This can be seen in the well publicised problems of a number of microcomputer manufacturers which, in turn, have engendered a definite degree of marketing caution — most of the newer products have been revamped versions of already successful machines, such as the BBC B+, the Commodore 128 and the Atari 130XE.

However, 1985 may also be noted as the year Atari emerged from the doldrums to re-establish itself as a major home micro manufacturer. This is, in part, due to the influence of the company's new owner, Jack Tramiel, who not only brought with him a shrewd business acumen, but also generated, as one of the 'characters' of the industry, a great deal of media attention.

On its own, this may not have been enough to save Atari, since its major problem was its product line, which was seen as essentially out of date. The launch of the Atari 520ST dramatically changes that situation. The computer is designed around the 16-bit Motorola 68000 processor — the same one used in the Apple Macintosh. This chip is generally regarded as the most advanced currently available in large numbers, with large-scale 32-bit internal architecture, 17 32-bit registers, a 16-bit data bus and a 24-bit address bus. Obviously, a very powerful computer could be designed around these specifications, which is precisely what Atari has done.

Produced in the same style and livery as the 130XE (see page 1309), the 520ST is finished in strong grey plastic. The keyboard is divided into four sections, above which are 10 programmable function keys — set into the casing like those on the 130XE. Below these is a standard keyboard, with the addition of an Alternate key. This is used, when the mouse is not attached, to control the screen cursor.

To the right is a cursor cluster, together with other keys for the full screen editor, such as Clear and Insert. Just above the cursor cluster are the Help and Undo keys, and on the far right of the keyboard is a numeric keypad with keys for the arithmetic functions.

As a departure from previous practice, Atari has adopted a number of 'standard' interfaces for the computer — previous Atari machines have had their own serial ports for all peripherals. The Atari 520ST has a Centronics parallel interface to connect to printers and a 25-way D-type RS232 serial port to fit to a modem or other serial device.

Most interestingly, the computer has been equipped with a MIDI interface. This is fitted as a

**Jack In The Box**
The 520ST represents Atari's attempt to regain the share it held in the microcomputer market in the late 1970s. Based around the Motorola 68000 processor, the computer has a number of advanced features — such as 512 Kbytes of RAM, MIDI ports and the GEM operating system. Hitherto, GEM has been available only on much more expensive machines

pair of DIN sockets (for MIDI IN and MIDI OUT), which means that the computer can directly control the operation of a number of synthesisers and other musical instruments. However, the MIDI ports have other uses. Because MIDI is a fast (31.25 Kbaud) serial device, it can also be used as an alternative method of transmitting data between computers. Atari intends to take advantage of this facility to produce a local area network (LAN; see page 969) system using the MIDI ports.

Another interesting facility that has been added to the new computer is a 'hard disk' interface. Atari is currently hoping to be among the first to launch a 'CD-ROM' player. This is a development of the compact disk that will enable up to 800 Mbytes of information to be stored on a single disk. Of course, at present, compact disks are read-only devices, but their potential is phenomenal.

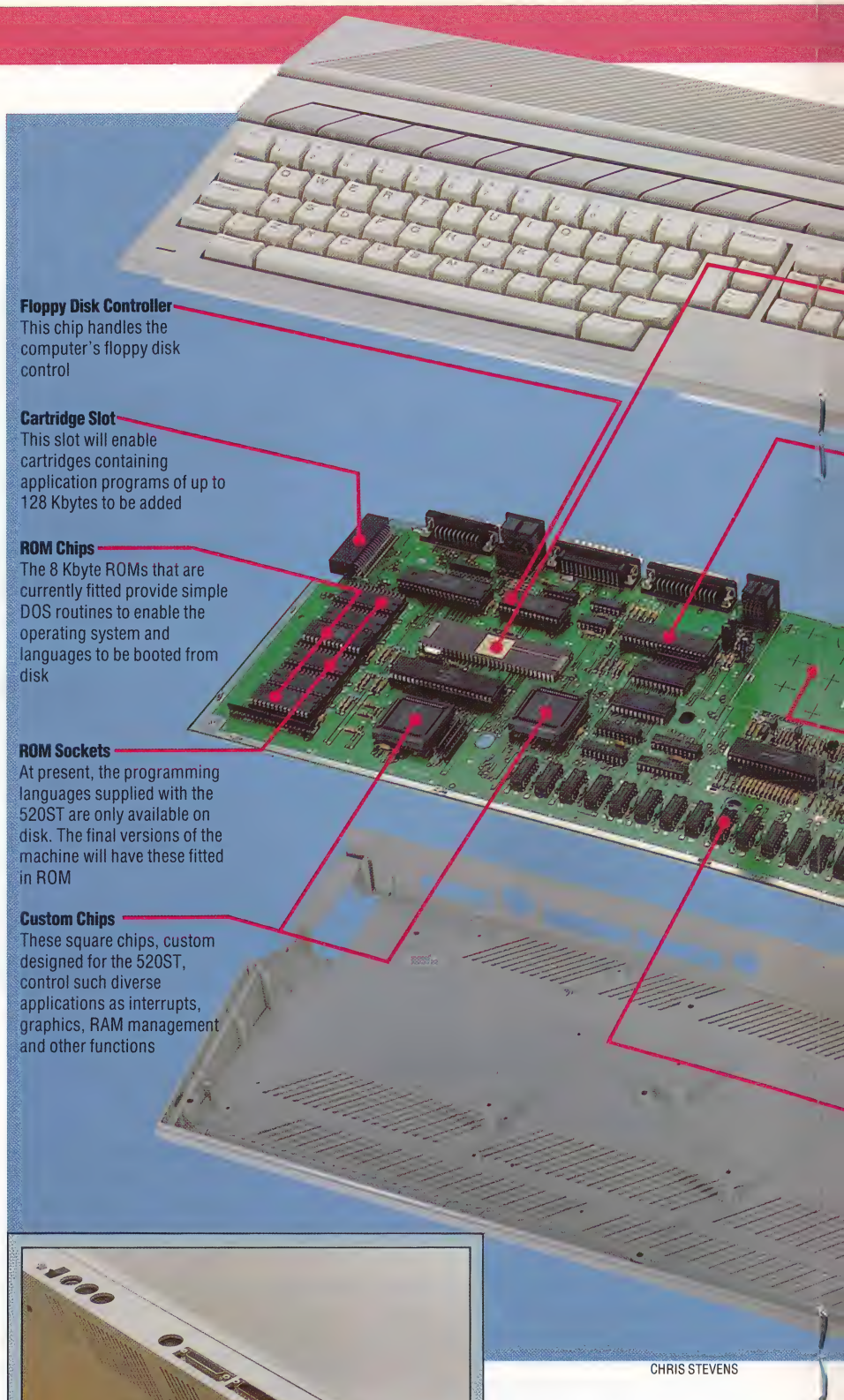
Included in the price of the new Atari machine is a disk drive, which plugs into the single floppy disk interface. If another disk is required, it can be plugged into the first disk drive. No cassette port is provided. Atari has opted for the Sony 3½in disks, which are becoming increasingly popular on a wide variety of micros. The disk drives are being produced in single- or double-sided versions which, when formatted, provide 320 Kbytes on each side.

To complete the list of interfaces provided on the computer, there is a power socket, RGB monitor, and RF ports (on future versions of the machine), a cartridge port capable of using 128 Kbyte ROMs, and a pair of joystick ports on the right-hand side; although these are not intended specifically for joysticks, but, rather, for mouse controllers.

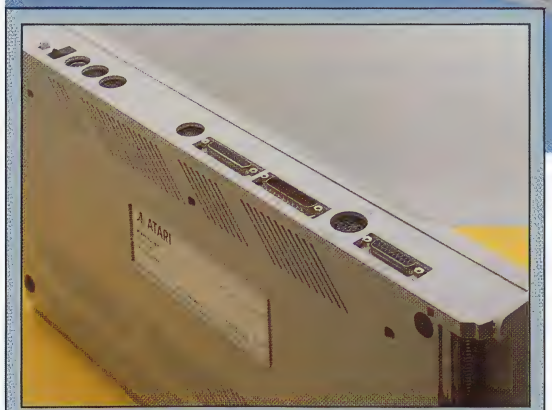
## GEM ENVIRONMENT

It is this last feature that indicates the real reason for the excitement that has been generated by this computer. The 520ST is the first low-cost micro to be introduced with the GEM environment as the standard front-end for the operating system. Developed by Digital Research, GEM (an acronym for Graphics Environment Manager) provides a WIMP (Windows, Icons, Mouse Program) system, like the one that has proved so successful on the Apple Macintosh. In fact, the GEM system as implemented on the new Atari micro bears a striking similarity to the Macintosh.

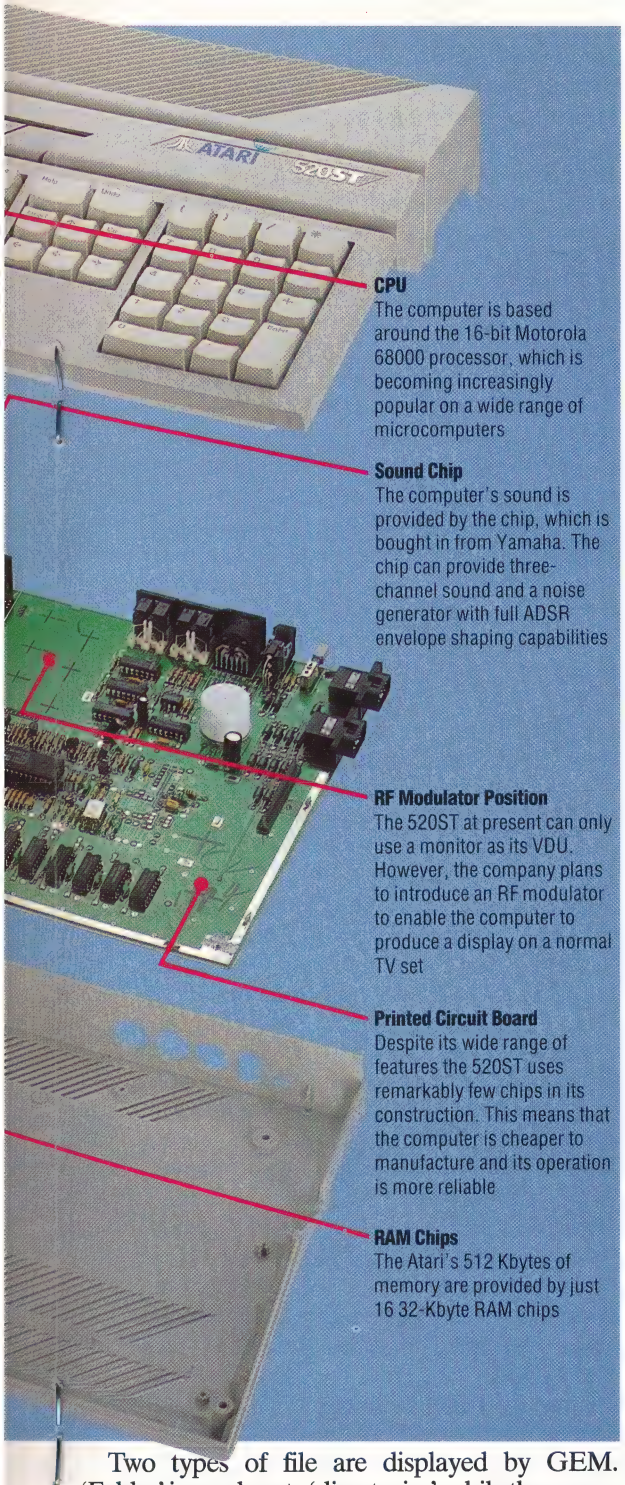
Control of the system is provided by moving an arrow cursor (which converts to a 'bee' icon when the machine is 'busy') around the screen to a number of icons. Pressing the button on the mouse selects that icon. Also available, at the top of the screen, are a number of menus that can be 'pulled down' as windows. A disk drive, for example, is selected by choosing the 'filing cabinet' icon in the top left-hand corner of the initial screen. Then, various file manipulations can be performed by pulling down the 'files' menu; selecting 'options' displays the directory.


Floppy Disk Controller
This chip handles the computer's floppy disk control
Cartridge Slot
This slot will enable cartridges containing application programs of up to 128 Kbytes to be added
ROM Chips
The 8 Kbyte ROMs that are currently fitted provide simple DOS routines to enable the operating system and languages to be booted from disk
ROM Sockets
At present, the programming languages supplied with the 520ST are only available on disk. The final versions of the machine will have these fitted in ROM
Custom Chips
These square chips, custom designed for the 520ST, control such diverse applications as interrupts, graphics, RAM management and other functions

CHRIS STEVENS



**Something For Everyone**
Atari have abandoned the system whereby its computers could only be fitted to the company's own peripherals and have fitted a number of 'standard' interfaces to the 520ST. The range of ports makes it one of the best equipped machines now on the market. Among these are RS232, Centronics and MIDI ports, as well as a serial disk interface and RGB monitor socket


ATARI 520ST
CPU
The computer is based around the 16-bit Motorola 68000 processor, which is becoming increasingly popular on a wide range of microcomputers
Sound Chip
The computer's sound is provided by the chip, which is bought in from Yamaha. The chip can provide three-channel sound and a noise generator with full ADSR envelope shaping capabilities
RF Modulator Position
The 520ST at present can only use a monitor as its VDU. However, the company plans to introduce an RF modulator to enable the computer to produce a display on a normal TV set
Printed Circuit Board
Despite its wide range of features the 520ST uses remarkably few chips in its construction. This means that the computer is cheaper to manufacture and its operation is more reliable
RAM Chips
The Atari's 512 Kbytes of memory are provided by just 16 32-Kbyte RAM chips

Two types of file are displayed by GEM. 'Folder' icons denote 'directories', while the square icons represent program files. Simply moving the cursor to the file will LOAD and RUN the program. Should you wish to delete a file, you only have to move the relevant icon to the 'wastepaper bin' icon. The ease with which even a novice can learn to use GEM is remarkably evident to anyone who has used some of the more conventional operating systems, such as CP/M and MS-DOS.

There are three graphics modes available for the 520ST — high-, medium- and low-resolution. The highest mode has a resolution of 640 by 400 pixels, but only appears in monochrome, which gives the machine an even more Mac-like appearance. The lower resolution modes demonstrate the machine's colour capabilities, but many users may prefer the high-resolution monochrome mode. Atari is providing a choice of either a monochrome monitor or a colour model for the 520ST.

The GEM 'desktop' environment (so called because you move the icons around in much the same way as you would move papers on a desk), runs under TOS — Tramiel Operating System. This is a custom-built version of Digital Research's CP/M 68 operating system, itself a development of Digital Research's popular eight-bit OS, which has been redesigned for 68000-based machines. However, unlike CP/M 68, the operating system allows files to be organised into 'directories' rather like MS-DOS, the *de facto* standard 16-bit operating system.

Bundled with the 520ST are a number of applications programs. First, there is GEM Write, a word processing package, and GEM Paint, which, as the name suggests, is a graphics design package. Similar to the Macintosh equivalents MacPaint and MacDraw, the package provides for a number of 'brush' sizes and 'fill' designs, as well as a zoom facility that enables the user to focus on a small area of the screen in order to produce particularly fine work. While GEM Paint lacks some of the facilities of the Macintosh, it is nevertheless a very sophisticated program.

## ROM-BASED LANGUAGES

Also included with the computer are the programming languages ST-BASIC and ST-LOGO, which, on early machines, are provided on disk but in later versions should appear in ROM on board. Like TOS, the languages are developments of previous Digital Research products — the BASIC is DR's own version of Microsoft BASIC, known as Personal BASIC, while the LOGO is a version of the popular Dr LOGO, now included with a number of machines. The languages have been adapted to take advantage of the windowing capabilities of the GEM environment.

The Atari 520ST is undoubtedly an important machine — not only for the well-being of the Atari Corporation itself, but also for the 'budget' computer market. The complete system, costing £750, includes a disk drive, a monochrome monitor and a mouse, as well as the bundled software. This puts it within reach of the 'serious' home user and the small business, although it is perhaps too expensive to generate very large sales in the UK.

Together with the GEM packages and 512 Kbytes of memory, the user obtains a configuration similar to the apple 'Fat Mac' at less than half the price. Should a smaller version of the computer appear, with, for example, 256 Kbytes of memory, the resultant sales would be far greater. However, one thing is certain: when the 520ST becomes available in sufficient quantities to make an impact on the high streets, the budget computer market will have moved into a new era.

## ATARI 520ST

**PRICE**
£749.99 inc. VAT

**DIMENSIONS**
470×240×60mm

**CPU**
Motorola 68000 running at 8MHz

**MEMORY**
512 Kbytes RAM

**SCREEN**
Three resolution modes; highest is monochrome mode at 640×400 pixels

**INTERFACES**
MIDI (two DIN sockets), Centronics parallel interface, RS232 serial port, floppy disk interface, power socket, RGB monitor, RF, cartridge and joystick ports. There is also a 'hard disk' interface for future CD-ROM players

**DISK DRIVE**
Single double-side Sony 3½in disk drive; each side provides 320 Kbytes

**OPERATING SYSTEM**
TOS and GEM as standard
LANGUAGES AVAILABLE
ST-BASIC and ST-LOGO

**KEYBOARD**
84 keys plus 10 function keys

**DOCUMENTATION**
Owner's manual is well laid out with an ample number of diagrams and screen shots accompanying the concise text

**STRENGTHS**
The computer offers 16-bit technology, large user memory and a user-friendly operating system that has previously been available only on much more expensive machines

**WEAKNESSES**
Despite its impressive specifications, the system has yet to prove itself, and the computer is still perhaps too expensive to generate volume sales on which the vital software support depends

# POINT TO POINT

**The DVM module at the heart of our digital multimeter has a basic sensitivity of 1.9999v. However, to be useful, it needs an input attenuator to measure voltages higher than 2v, plus special circuitry to measure amps and ohms. We now begin the meter's construction by considering the layout of the passive components.**

## Parts List

| No | Ref | Item | Part Number |
|---|---|---|---|
| 1 | C1 | 0.47uF polypropylene capacitor | RS 114-626 (1) |
| 2 | C2-3 | 1.0uF polycarbonate capacitor | Maplin WW53H |
| 1 | C4 | 0.1uF polycarbonate capacitor | Maplin WW41U |
| 1 | C5 | 470pF polystyrene capacitor | Maplin BX32K |
| 5 | C6 | 22,000pF disc ceramic capacitor | Maplin BX01B |
| 1 | D1 | 9491 band-gap 1.22v ref diode | RS283-283 (2) |
| 1 | D2 | 1N4148 diode | Maplin QL80B |
| 6 | TR1-5, TR7 | 2N2219 NPN transistor | Maplin QR11M |
| 1 | TR6 | 2N2905 PNP transistor | Maplin QR17T |
| 1 | VR1 | 10k multi-turn cermet variable resistor | Maplin WR49D |
| 1 | VR2 | 10k cermet preset potentiometer | Maplin WR42V |
| 1 | IC1 | 7135 A/D converter chip | RS 302-261 (3) |
| 1 | IC2 | ICM7555 low power timer | Maplin YH63T |
| 1 | IC3 | 7447A 7-segment driver | Maplin QX55K |
| 1 | — | 28-pin DIL socket | Maplin BL21X |
| 2 | — | 16-pin DIL socket (4) | Maplin BL19V |
| 1 | R1 | 8.2K-ohm resistor (5) | |
| 7 | R2-8, R14, R17 | 180-ohm resistor | |
| 1 | R9 | 47K-ohm resistor | |
| 3 | R10-12 | 100K-ohm resistor | |
| 1 | R13, R15, R16 | 4.7K-ohm resistor | |

**Notes:**

1) Maplin's polycarbonate WW49D capacitor may be substituted (with some loss of performance)

2) If a 9491 cannot be obtained, Maplin's QH00A may be substituted (with some loss of performance)

3) This component can also be obtained from Trident Microsystems Ltd, Trident House, 53 Ormside Way, Redhill, Surrey RH1 2LS. Tel: (0737) 65900

4) Maplin does not supply the 8-pin DIL socket needed by the timer chip. However, a second 16-pin DIL socket can be used to mount this chip

5) Carbon or metal film 5% tolerance resistors should be used and are readily available from many sources

We recommend using 0.1in matrix board rather than Veroboard construction of the basic module. This is because Veroboard commits the wiring layout once and for all, whereas the point-to-point wiring of a matrix board allows slight changes to be made to the positioning of the wires to optimise performance.

Unlike ordinary digital circuits, an A/D converter has very sensitive analogue inputs (high input impedance). This means that stray and unwanted signals can leak into the analogue inputs unless special steps are taken. Our design keeps all analogue circuitry to one side of the A/D converter chip (pins 1 to 10), and all digital circuitry to the other. Even so, when building the prototype, we found that slight changes in the position of the wiring could significantly improve the performance of the circuit. Ideally, such a circuit would use a tried and tested printed circuit board, but making a PCB is beyond the resources of the average constructor, so we have opted for point-to-point wiring.

The wiring diagram shows a top view of the circuit board; the interconnecting wires would be on the under-side of the board. The wiring layout is very stylised to make it clearer, with only a few wires shown connected directly from one point to another.

This angular layout is merely for clarity, and need not be followed exactly. Although it might look neater, it is better not to wire tightly from point to point as some slack in the wires allows them to be moved later to optimise the overall performance.

The prototype board was wired using wire-wrapping sockets and a special wire-wrapping tool to connect them. There is a lot to be said for this technique, because if you connect a wire incorrectly, it can be unwrapped and reconnected with ease. The alternative to this method is to use ordinary thin interconnecting wire and a soldering iron.

Most semiconductor chips have pins with 0.1in (2.45mm) centres, so the most suitable circuit board is an 0.1in matrix board with each hole surrounded with a ring of copper for soldering purposes. Maplin offers a copperless version of this type of board (see Parts List), but the copper-clad type is undoubtedly easier to work with.

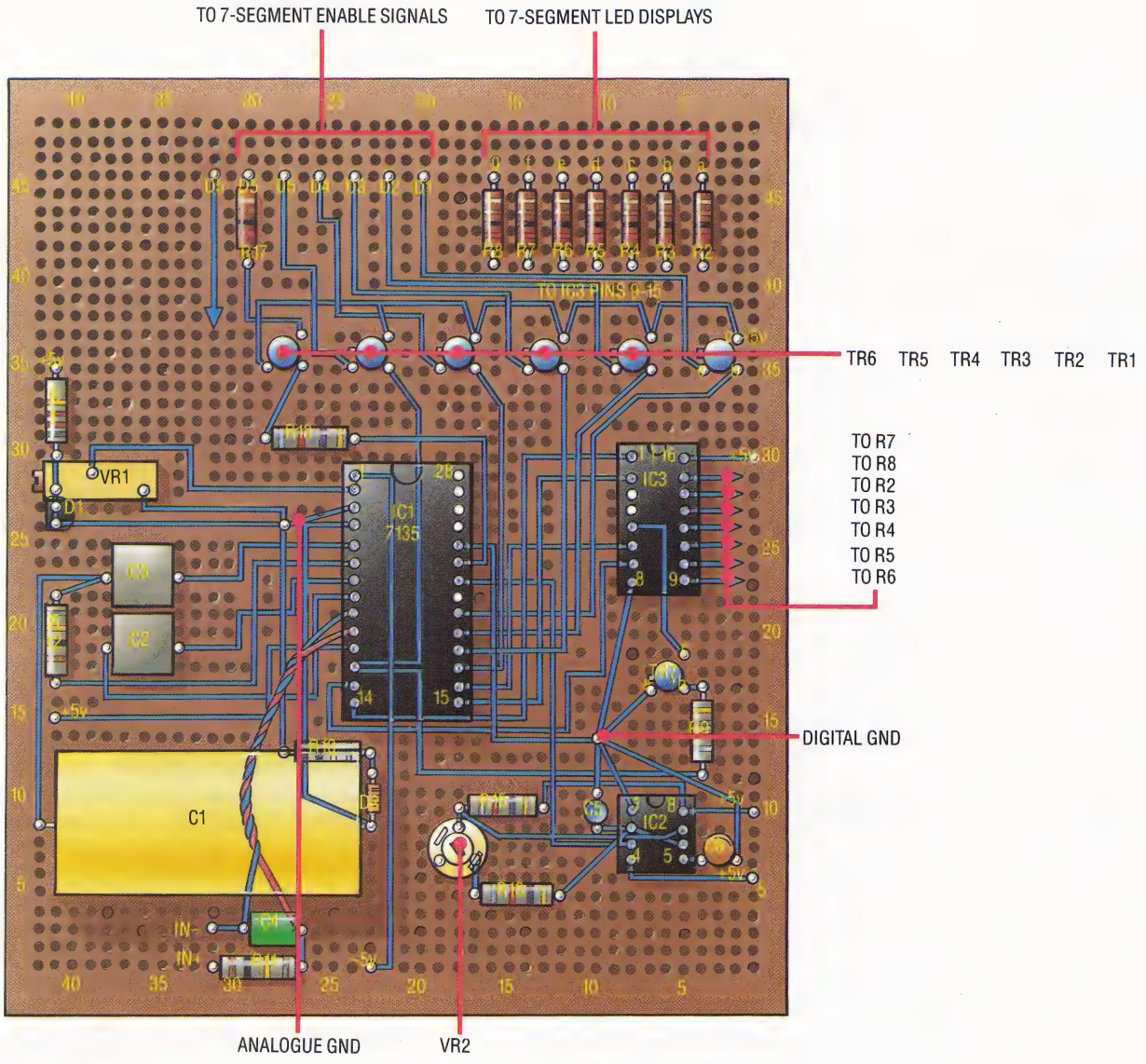
To allow the components plenty of room, the matrix board should have about 50 rows and 45 columns of holes. Start by inserting the three IC sockets — the 28-pin DIL socket for the 7135 A/D converter (ICI), an eight-pin socket for the 7555 timer chip (IC2) and a 16-pin socket for the 7447A seven-segment driver (IC3). If you are using a copper-clad matrix board, solder two pins on opposite corners of each DIP socket to the back of the board. If you are using the copperless type of matrix board, a small blob of solder on the corner pins will hold the sockets in place.

Before attaching any other components, solder or wire-wrap the +5v lines (Vcc) to the three ICs. For clarity, the complete Vcc rail is not shown, but eventually all the +5v points (such as pin 16 on IC3) will be connected together and fed by the power supply. Only one chip, the 7135, needs a −5v (Vee) supply. This goes from pin 1 of IC1 to a suitable point on the edge of the DVM board, as shown.

## GROUND POINTS

Two separate ground points are needed: one for the analogue ground (located near pin 4 of IC1) and one for the digital ground. As we have stressed before (see page 1516), it is important to wire all analogue grounds to a single analogue ground point, and all digital grounds to another single digital ground point. Connect the analogue ground of IC1 (pin 3) to the analogue ground and the grounds of IC2 and IC3 (pins 1 and 8 respectively) to the digital ground. The digital ground of IC1 (pin 24) is also connected to the digital ground point.

# Digital Multimeter Layout




KEVIN JONES

## Component Mounting

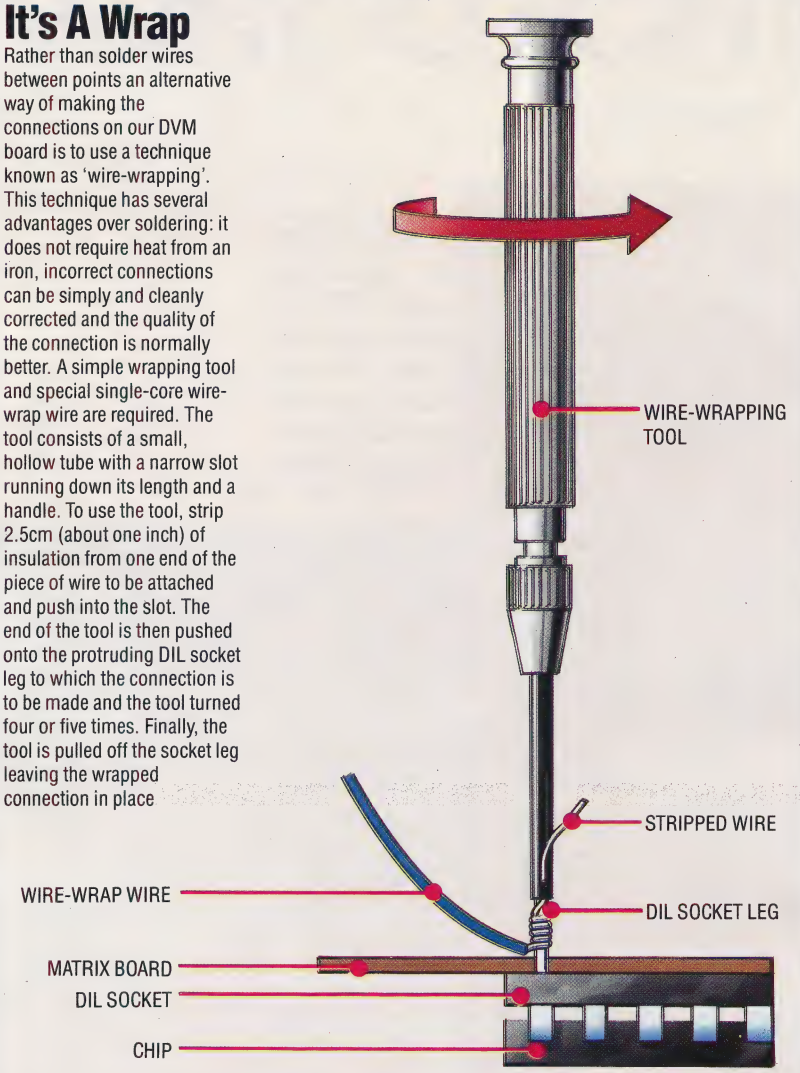
The layout diagram shows the components mounted on 0.1in matrix board. The point-to-point wiring is shown in an angular fashion to aid clarity. In practice, the wiring between points on the board should be slack enough to allow the wires to be moved as the position of the wiring can affect performance of the meter. Note that connections between analogue and digital ground points, D5, to digital ground and all the +5v power feeds are not shown explicitly.

The connection between analogue and digital grounds should be made with fairly thick copper wire. The simplest method of providing the +5v feeds is to connect all these connections to a single piece of wire that runs around the edge of the board. As mentioned in the main text, if voltages to be measured are relative to ground, pin 9 on the 7135 chip should be tied to the analogue ground point. Solder all passive components in place, as shown, but do not yet mount the chips in their sockets as they are easily damaged. We shall deal with this part of the construction in the next instalment.

22,000 pF disc capacitors should also be connected between the chip power supplies and ground. On IC1 connect a capacitor between pin 1 and the analogue ground point and another between pin 11 and analogue ground. On IC2 connect a capacitor between pin 8 and digital ground and on IC3 between pin 16 and digital ground. A 47 µF electrolytic capacitor should also be connected between pin 1 of IC1 and analogue ground. Note that with this type of capacitor the orientation is important. Ensure that you connect the negative leg of the capacitor to pin 1 of the chip and the positive side to ground

## It's A Wrap

Rather than solder wires between points an alternative way of making the connections on our DVM board is to use a technique known as 'wire-wrapping'. This technique has several advantages over soldering: it does not require heat from an iron, incorrect connections can be simply and cleanly corrected and the quality of the connection is normally better. A simple wrapping tool and special single-core wire-wrap wire are required. The tool consists of a small, hollow tube with a narrow slot running down its length and a handle. To use the tool, strip 2.5cm (about one inch) of insulation from one end of the piece of wire to be attached and push into the slot. The end of the tool is then pushed onto the protruding DIL socket leg to which the connection is to be made and the tool turned four or five times. Finally, the tool is pulled off the socket leg leaving the wrapped connection in place




Finally, connect the two ground points together using fairly stout wire. (For clarity, this connection is not illustrated, but don't forget to do it!)

Before installing any other components, use a voltmeter or a simple continuity tester to check for continuity from the power supply entry point to the Vcc pins on the three ICs. Similarly, check for continuity between all the ground points.

Having installed the IC sockets, the other components can be inserted in the positions shown. The exact sizes shown are for the parts specified in the Parts List. Most of the components can be substituted by similar items, but the dimensions may be different and you may have to alter the layout slightly. The component most difficult to get is C1 — the 0.47µF integrating capacitor.

For best performance, this should be a polypropylene type, such as Radio Spares Components' part number 114-626. None of the retailers we checked stock polypropylene capacitors, and Maplin doesn't list them. Either substitute a polycarbonate capacitor (which will work, but less well), or find an electronics retailer who is willing to order RS parts for you. RS Components sell only to the trade and industry; they refuse to deal with the public.

Another problematic part has proved to be the seven-segment LEDs. Maplin offers double units (either two '8s', or a '+/−1' and an '8'), or individual '8s', but no individual '+/−1s'. This means that, using Maplin parts, it is impossible to build a $4\frac{1}{2}$-digit display able to show a true plus and minus sign without having a redundant digit, since their double LEDs and their single ones are different sizes.

RS Components do have suitable LEDs (used in the prototype) but in view of the difficulty and expense of obtaining RS parts, we have modified the original circuit slightly to work with Maplin single digit, 0.5in LEDs. This uses an ordinary seven-segment LED to display a blank or a 1, and also to display a minus sign using the middle horizontal segment (segment G) when the signal being measured is negative. Positive signals will be displayed with no sign in front of them.

VR1 should be a good quality multi-turn trimpot; VR2 can be any preset trimmer. D1 should ideally be a band gap reference diode such as a type 9491 (available from RS as part number 283-283). To avoid the inconvenience of obtaining this part, a 2.7v zener diode can be substituted (such as BZY88C2V7 — Maplin part number QH00A). Again, this part will work, but at a reduced overall performance.

## TRANSISTOR TYPES

The transistors (apart from TR7) are used to switch on the LED displays. Note that transistors 1 to 5 are NPN types and TR6 is a PNP type. This is because TR6 has to switch on in response to a logic LO signal, while the others have to switch on in response to a logic HI signal. TR7 is used in the zero blanking circuit. D2 can be any small silicon diode, though the part specified here is readily available.

Connect the input points to IC1 pins 9 and 10, using a short length of twisted wire or some thin coaxial cable (braid to pin 9; core to pin 10). On our prototype, very much better results were obtained by tying IC1 pin 9 to the analogue gound (not shown in the diagram). Making this connection is recommended if a floating point input is not required, as voltages are measured against ground rather than the difference between positive and negative voltages.

Before even attempting to insert the ICs, check your wiring thoroughly against the wiring diagram. Note, for example, that TR6 has no lug and that the collector and emitter are connected in the opposite sense to those of TR1 and TR5.

In the next instalment, we shall deal with the wiring of the LED panel and the handling of the ICs. Careless handling will destroy ICs very quickly, so it is recommended that you follow the instructions carefully, unless you are totally familiar with handling CMOS components.

# ARRAY OF INTELLIGENCE

**FORTH has no standard provision for array structures, but you can still, because of the language's extendability, define the customary facilities for yourself. In seeing how this is done, we also present a unique feature of FORTH that is useful for far more than arrays.**

We've already seen some of the ideas that are needed to be combined to define our own arrays when we used CREATE in the Eratosthenes' Sieve program (see page 1504); it was equivalent to:

```
CREATE BITS 8192 ALLOT
```

In that program, CREATE encloses a header in the dictionary using the name BITS from the keyboard, and it also encloses a code field that makes BITS, when you use it, leave its parameter field address, pfa, on the stack. However, CREATE itself does nothing about setting up the parameter field, and this is where ALLOT comes in. ALLOT takes a number off the stack and encloses space for the corresponding number of bytes in the dictionary. Since in this case they are enclosed immediately after the code field enclosed by CREATE, they constitute the parameter field for BITS.

## POWERS OF THREE

Here is a simple example that defines a word, 3**, to calculate powers of three. To make it quick, it uses an array containing all the possible answers, and it's set up using CREATE with the FORTH word , (just a comma):

```
VARIABLE MAXINDEX
: POWERS, ( - - )
      ( encloses all powers of three that can fit in
      two bytes, and puts index of biggest power in
      MAXINDEX. )
   -1 MAXINDEX !
   1          ( smallest power )
   BEGIN
      DUP , ( enclose power )
      1   MAXINDEX +!
      3   UM* ( next power, double length )
            ( replace UM* by U* in FORTH-79 and
            figFORTH )
   UNTIL   ( until next power takes more than 2
            bytes )
   DROP    ( least significant bytes of double length
            power )
;

CREATE 3POWERS    POWERS,
            ( no need for ALLOT. , does the
            enclosing )
```

## In Close Array

Defining a pseudo-array structure in FORTH is not difficult, thanks to CREATE and DOES>. The diagram shows, step by step, FORTH's response to the input 11 1ARRAY RUNS where 1ARRAY has previously been enclosed in the dictionary as a colon definition. The effect on the stack is also shown

| Colon Definition | 11 1ARRAY RUNS | Stack |
|---|---|---|
| : 1ARRAY | CREATE takes RUNS, allots it a name field and a code field. RUNS will now | 11 |
| CREATE | leave its pfa on the stack when used | |
| DUP + | DUP + multiplies 11 by two and puts the result on the stack | 22 |
| HERE | HERE puts the next available address in the dictionary on the stack. Since we have just enclosed RUNS, this is the pfa of RUNS | PFA<br>22 |
| OVER | OVER copies 22 onto the top of the stack, above the pfa of RUNS | 22<br>PFA<br>22 |
| ALLOT | ALLOT takes 22 off the stack, and encloses 22 bytes in the dictionary, which (since we have just enclosed RUNS) constitutes the parameter field for RUNS | PFA<br>22 |
| SWAP 0 FILL | SWAP puts 22 on top of the pfa of RUNS; 0 is placed on the stack; FILL then takes all three parameters and zeros all 22 bytes in the parameter field | 0<br>22<br>PFA → EMPTY |
| DOES> | RUN TIME (after DOES>) | |
| | 99 2 RUNS ! (for example) sets up the stack as shown and then stores the value 99 in element 2 of the array RUNS as follows: | PFA<br>2<br>99 |
| SWAP | SWAP puts the 'array subscript' (2) on top of the stack . . . | 2<br>PFA<br>99 |
| DUP + | DUP + doubles it . . . | 4<br>PFA<br>99 |
| + | + adds the subscript (which is effectively acting as an offset value) to | PFA<br>+OFF-SET |
| ; | the pfa of RUNS . . . | |
| ! | ! takes 99 and stores it at the correct address | EMPTY |

CAROLINE CLAYTON

```
: 3**     ( n -- 3**n )
MAXINDEX @ OVER U< IF
   DROP    0  ( n not between 0 and MAXINDEX.
              Leave result 0 )
ELSE          ( want memory contents at
              3POWERS + 2*n )
   2 * 3POWERS + @
THEN
;
```

3POWERS was a one-dimensional numeric array. You could equally well have set it up using ALLOT (remember that you have to ALLOT two bytes for each element in the array) and !. Character arrays are similar, but since each character represents only one byte, you must use C, , C! and C@ instead of , , ! and @.

## THE CREATE STRUCTURE

Every time you use an array like 3POWERS, you need to do something like:

```
2 * 3POWERS +
```

This is because 3POWERS doesn't know it's an array. It just leaves its parameter field address and lets you sort out what to do with it. To make more 'intelligent' words, there is a very interesting construction that uses CREATE (or <BUILDS in figFORTH) and DOES>. These let you define new words that are embellished versions of CREATE, so they are in fact new defining words. They do CREATE, but you can also tell them to do other things at the same time as the CREATE. They can also tell you what the newly CREATEd word is to do when you use it (instead of just stacking its parameter field address).

Here is an example that creates intelligent arrays:

```
: 1ARRAY    ( n -- )
      ( creates a 1-dimensional numeric array wit
      dimension n )
   CREATE     ( name is taken from after 1ARRAY
              when you use it )
   DUP +      ( a quick way of doing 2 * )
   HERE       ( remember parameter field address
              for FILL )
   OVER ALLOT
              ( now have 2*n, pfa )
   SWAP 0 FILL
              ( zero all bytes in parameter field )
DOES>         ( subscript, pfa -- address of
              element )
   SWAP DUP + ( pfa, 2*subscript )
   +
;
```

It falls into two parts: the define-time part, before DOES>, is performed when you use 1ARRAY and sets up the parameter field of the new word.

Suppose you say:

```
11 1ARRAY RUNS
```

This creates a new word, RUNS, and ALLOTs it a 22-byte parameter field initialised to zeros. The second part of 1ARRAY, after DOES> , is the run-time part. This is done when RUNS is used. RUNS still puts its parameter field address on the stack, but after that the run-time part of 1ARRAY applies its intelligence.

In effect you now have 11 variables, called:

```
0 RUNS
1 RUNS
:

:
10 RUNS
```

You use them just like ordinary variables, for instance:

```
99 2 RUNS !
4 RUNS @ .
```

## FURTHER APPLICATIONS

That was a fairly straightforward application of CREATE . . . DOES>. But once you get the hang of it, you will find that it can do far cleverer things. For instance, if you've started writing your own FORTH programs, you'll no doubt have forgotten many times to include @ when defining variables. With CREATE and DOES>, you can define a kind of variable that doesn't use @. Normally, it just leaves its value on the stack, like a constant, but a problem arises when you want to change its value. It's best then to use a word, —>, as in:

```
new value —> name of new style variable
```

—> does nothing except leave a 'note' (by setting a variable) saying that it has been used, and this tells the 'new-style' variable that instead of stacking its current value, it must get a new value from the stack.

```
VARIABLE —>USED
0 —>USED !
: —> —1 —>USED ! ;
```

We now want a replacement for the defining word VARIABLE. This (let's call it VAR) is a new defining word, using CREATE . . . DOES>:

```
: VAR ( -- )
   CREATE
   0 , ( allot 2 bytes, initialised to 0 )
DOES>
   —>USED @ IF ( new value, pfa -- )
      !          ( set new value )
      0 —>USED !
   ELSE          ( pfa -- current value )
      @
   THEN
;
```

Here's how to define a VAR variable x, set it to 99, and then print its value:

```
VAR x
99 —> x
x .
```

The reason CREATE and DOES> are so useful is that they let you combine two properties usually kept separate. Most programming languages have data, which is 'passive' and requires acting upon, and routines, which are 'active' but need to data. With CREATE and DOES>, you can set up intelligent data that will act on itself by using a DOES> run-time part. This idea is very fruitful, but to be able to use it you must forget the distinction between data and routines that almost all other languages enforce.

## Useful Words

**, ( n – – )** Encloses n as two bytes in the dictionary. There is a one-byte version C,.

**HERE ( – – address )** This stacks the address of the place in the dictionary where the next enclosure will be made.

**FILL ( address, n, filler – – )** This fills in n bytes, starting at the given address, with the given filler byte.

## Array Of Arrays

The arrays defined with 1ARRAY are only 'mildly' intelligent. They work quite quickly, but they don't check that the subscript you give them is reasonable and there is a risk that you can accidentally overwrite something outside the array itself. If 1ARRAY stored the dimension at the beginning of the parameter field, then the run-time part could check that the subscript supplied is less than the dimension, and complain if not.

Some languages, like PASCAL and BASIC, check array subscripts like this, while others, like C, go for speed and don't check. FORTH forces you to work out what you really want and also to do it accordingly.

Languages also differ in whether they use subscript 0 for the first element or subscript 1 — or indeed any other subscript. Again, in FORTH, you can tailor the content of arrays to suit your own taste.

Multi-dimensional arrays are similar in principle, but more complicated. You normally imagine a multi-dimensional array as a rectangle or a three-dimensional block (or worse), but the computer stores it row by row in a long list. You convert your several subscripts into an offset into this list by a process of multiplying by the dimensions.

For instance, for a three-dimensional array with dimensions 2, 3 and 4 (24 elements altogether), you convert subscripts i, j and k into an offset:

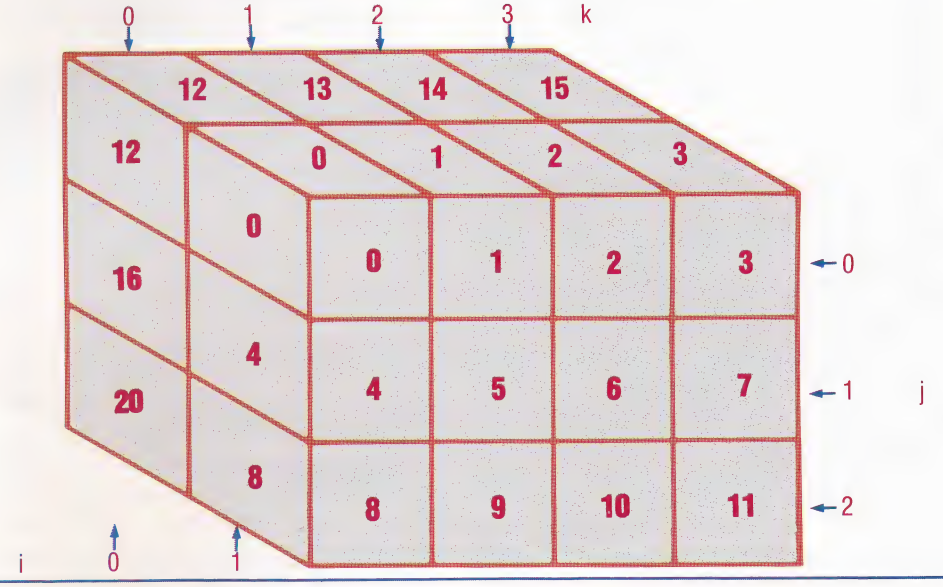
```
(i*3 + j)*4 + k
```

Here is how you'd define 3ARRAY:

```
: 3ARRAY ( d, e, f – – )
    ( creates a 3-dimensional array with
     dimensions d, e and f )
  CREATE
    OVER , DUP ,     ( enclose e and f )
    HERE             ( d, e, f, pfa+4 )
    2 ROT * ROT * ROT * ( pfa+4, 2*d*e*f )
    DUP ALLOT        ( enclose 2*d*e*f bytes )
    0 FILL           ( initialise them to 0 )
  DOES>              ( i, j, k, pfa – – address )
    DUP @            ( i, j, k, pfa, e )
    4 ROLL *         ( j, k, pfa, i*e )
    3 ROLL +         ( k, pfa, i*e+j )
    OVER 2+ @        ( k, pfa, i*e+j, f )
    *                ( k, pfa, [i*e+j]*f )
    ROT + DUP +      ( pfa, 2*[[i*e+j]*f+k] )
    SWAP 4 + +
;
```

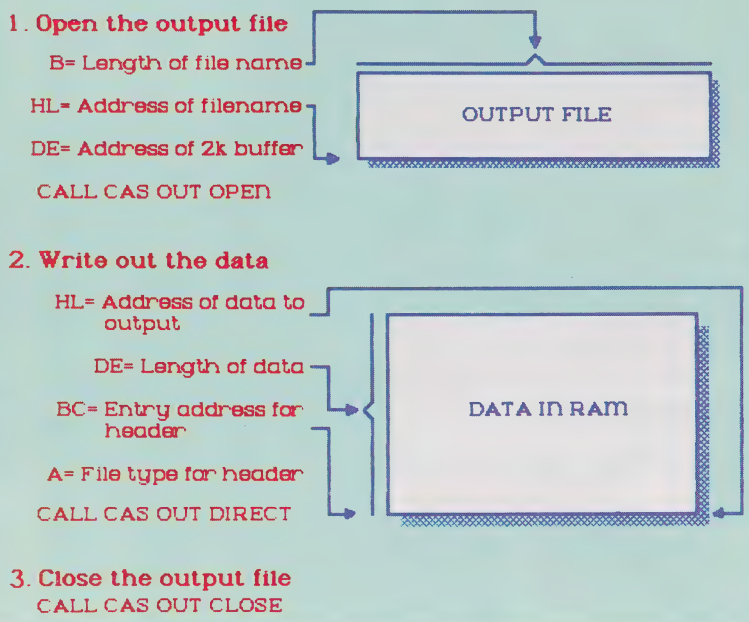
(Note that in FORTH-79, the number before ROLL must be one bigger.)

It is possible to write analogous words (4ARRAY, 5ARRAY and so on) along similar lines. If this seems complicated, the best way to understand it is to see how C does it. It is even possible to write a defining word, NARRAY, that takes a number off the stack telling it how many dimensions there are below



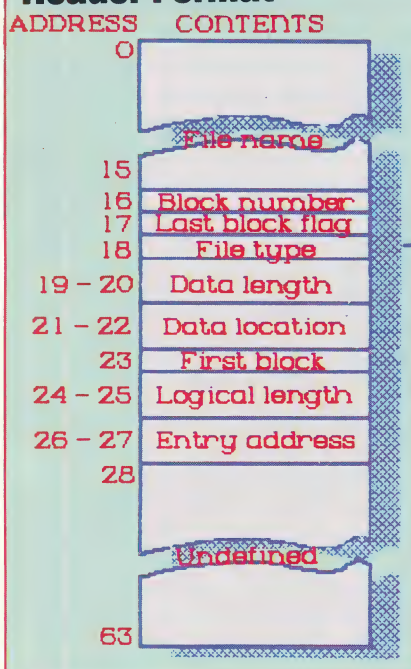
## Writing A Block Of Memory To A File

The following procedure should be adopted when writing memory to tape or disk:


1. Open the output file
B= Length of file name
HL= Address of filename
DE= Address of 2k buffer
CALL CAS OUT OPEN
OUTPUT FILE
2. Write out the data
HL= Address of data to output
DE= Length of data
BC= Entry address for header
A= File type for header
CALL CAS OUT DIRECT
DATA IN RAM
3. Close the output file
CALL CAS OUT CLOSE

CAROLINE CLAYTON

## Header Format




Opening a file for input using CAS IN OPEN results in the file header being written to RAM and its address returned in HL. In addition, DE will contain the original location of the data (as indicated by the header), BC will hold the logical file length, and A the file type. Information held in the file type byte is bit-significant, as shown here

File type is defined as several different fields:

| Bit 0 | Bits 1 – 3 | Bits 4 – 7 |
|---|---|---|
| Protection | File contents | Version |
| 0= Protected<br>1= Unprotected | 0= BASIC<br>2= Binary<br>4= Screen<br>6= ASCII<br>8= Unallocated | 16= ASCII<br>32= Unallocated |

CAROLINE CLAYTON

# SAFE KEEPING

**The integral cassette deck in the Amstrad CPC 464 and disk drive in the CPC 664 make file transfer very efficient on these machines. We look at several of the operating system features for accessing the storage media that are available to the Amstrad machine code programmer.**

One of the major selling points of the Amstrad CPC 464 was its integral cassette deck, which allowed extremely reliable storage even at its faster data transfer rates. Now the CPC 664 has gone one stage further by replacing the cassette deck with a disk drive, giving even faster access times. Obviously, the hardware is of no use unless there is adequate software to support it; so what provision is made in the firmware for a machine code programmer to use these storage facilities?

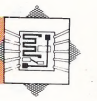
An analysis of firmware features can be divided into three areas: features that are general to file transfer, those that are exclusive to the tape system and those exclusive to the disk version.

Generally, each file contains two separate sections of information: the first is the header section and the second is the data. The header contains all the information that the firmware requires to determine how the file should be handled — for example, the type of file it is, where it should be loaded and how long it is. The data is simply the contents of the file. The firmware provides the user with all the information about the file, and the facility to manipulate it in memory.

The section of the firmware we are concerned with here is the cassette manager, although this term is a little misleading as the same entries are used to handle disk files as well, which we shall be looking at later.

The cassette manager uses two 'streams', one for input and one for output, both of which may be active at the same time. This feature is not of great use on a tape machine, but is a very powerful facility when used correctly with a disk drive. The streams have to be initialised before they can be accessed. Initialisation takes the form of assigning a name and a workspace area (buffer) to the relevant stream. Only one output and one input file can be active at a time, so if the output stream is unused it may not be defined as a second input stream. The entries dealing with file transfer are listed in the Data Storage Routines table.

Data can be transferred to or from the file in one of two modes: *character transfer* or *block transfer*. In the character transfer mode the firmware uses the buffer assigned to the stream to store the data temporarily. This means that access to or from the



tape (or disk) can be carried out in large blocks, while the program deals with the data in single byte stages. Obviously, a program that has to access the disk every time a character is required would be very slow. The two transfer modes are mutually exclusive — that is to say, once a file has been accessed in one mode it is impossible to switch and access it in the other.

In the block transfer mode (CAS IN DIRECT and CAS OUT DIRECT) the buffers assigned to the streams are not used; instead the data is directly read into or out of memory. The memory may be either in RAM or ROM.

The routines CAS IN CLOSE and CAS OUT CLOSE are especially important as they inform the firmware that the stream is now closed and therefore available for a new file. If a CAS OUT CLOSE is not performed following character output, the output buffer may be incomplete and so not yet written out. CAS OUT CLOSE avoids this problem by forcing the firmware to send any incomplete blocks of data.

Our diagram illustrates the process whereby a block of memory is written into a file. First of all, the file is opened using CAS OUT OPEN, which merely assigns a name to the file and reserves the output stream for that file's use. The data is then written out from memory using CAS OUT DIRECT, and finally the file is completed by calling CAS OUT CLOSE which, in this case, simply marks the output stream as available for use.

The cassette manager uses a data area, known as the 'header', to store information relevant to each individual file. In the case of a read operation, this information is taken from the start of the file and, in the case of a write, the header is set up in memory and then written out at the beginning of a file. The general layout of a header is shown in the Header Format diagram. Let's consider a few of the parameters involved.

At the firmware level, the *file type* has no effect — the file can be read from or written to regardless of its type. The type becomes significant when the firmware is being used by a higher level program, for example BASIC.

The *data location* is defined as the location from which the file was originally written. Note that this is only significant when the file is written directly from memory. When character output is used this represents the address of the output buffer. The

## Data Storage Routines

**Cassette System:**

| | |
|---|---|
| $BC65 | CAS INITIALISE<br>This routine ensures that all default conditions are set — i.e. input and output streams are closed and messages are enabled |
| $BC6B | CAS NOISY<br>Enables and disables the messages returned from the cassette manager |
| $BC77 | CAS IN OPEN<br>Initialises a file to read from |
| $BC80 | CAS IN CHAR<br>Reads a single character from the input file |
| $BC83 | CAS IN DIRECT<br>Reads an entire file into memory |
| $BC89 | CAS TEST EOF<br>Tests for end of input file |
| $BC7A | CAS IN CLOSE<br>Closes the current input file |
| $BC7D | CAS IN ABANDON<br>Closes the input file, even if it is not completely loaded |
| $BC8C | CAS OUT OPEN<br>Initialises a file to write to |
| $BC95 | CAS OUT CHAR<br>Sends a single character to the output file |
| $BC98 | CAS OUT DIRECT<br>Writes a block of memory to the output file |
| $BC8F | CAS OUT CLOSE<br>Closes a file on completion |
| $BC92 | CAS OUT ABANDON<br>Closes the file, discarding any unwritten information |
| $BC9B | CAS CATALOG<br>Generates a catalogue of files |

**Disk System:**

| | |
|---|---|
| $BE80 | SET MESSAGE<br>Enables and disables messages returned by disk |
| $BE83 | SETUP DISK<br>Initialises drive timings (step rate, load/unload delay and on/off timeout) |
| $BE86 | SELECT FORMAT<br>Sets the disk format |
| $BE89 | READ SECTOR<br>Reads a disk sector |
| $BE8C | WRITE SECTOR<br>Writes a disk sector |
| $BE8F | FORMAT TRACK<br>Formats a track |
| $BE92 | MOVE TRACK<br>Moves drive head to a given track |
| $BE95 | GET DR STATUS<br>Returns a byte indicating the status of the current drive |
| $BE98 | SET RETRY COUNT<br>Sets the number of retries on error |

Cassette and disk operations are carried out using these firmware calls, full details of which can be found in the Amstrad Firmware Manual (SOFT 158) published by Amsoft. Points to note about the Amstrad filing system are the ability to have both input and output files open simultaneously and the near-complete compatibility of disk and cassette file management syntax. Files can be read in and out directly from RAM (using CAS IN/OUT DIRECT), or character by character via a 2Kbyte buffer (CAS IN/OUT CHAR). However, once determined, the mode of input/output cannot be changed

*logical length* is the total number of data bytes in the file. Note that this does not include any header information, or any 'padding' that may have been required to fill the buffer.

The *entry address* is the address where machine code programs are started when auto-run from BASIC. The other fields are specific to cassette file-handling only, and are of limited use.

## ERROR DETECTION

Error detection is performed automatically by the firmware using a CRC code, although errors are rarely encountered thanks to the thoughtful system programming. Those errors that do occur are flagged by the firmware to allow the user to decide what action may be taken.

In order to facilitate the handling of read errors, cassette files are not written in their entirety, but are divided into two-Kbyte blocks instead. Each block is treated as a separate unit, and therefore a 'bad' block may be loaded again without having to reload any good blocks that may have preceeded it. This block approach necessarily slows down the speed of the file transfer, but the reliability achieved makes this a worthwhile sacrifice. The write speed to the cassette may be varied under software control using the CAS SET SPEED entry at $BC68, whereas the read speed is automatically selected by the firmware.

One of the most useful features of the Amstrad operating system, as far as the programmer is concerned, is the transparent installation of the disk commands. That is to say, all the general file routines have exactly the same parameters and so a program written for operation on cassette should work automatically with the disk drive without modification. This has been achieved by using the technique of jumpblock patching described in the previous instalment (see page 1524). However, the patch technique is not straightforward and a few useful hints may make its use less problematic.

Particular areas to watch are concerned with the use of memory. The disk system is controlled from an expansion ROM, although it does require a certain amount of RAM workspace as well. This is generally taken by default from the top of memory (but can be reassigned to other areas in RAM). Any program that wishes to use the disk must therefore be aware that this area of RAM ($504H bytes below HIMEM) is unavailable.

Another aspect, which is derived from the implementation of CP/M filename compatability, is that disk file names cannot be longer than 12 characters. It is, therefore, best to assume a length of eight characters to ensure that no problems arise when translating from tape to disk.

Disk headers behave in a slightly different manner to the tape headers; references to blocks no longer have a significance and so these are ignored. The header is also checksummed, to allow the firmware to distinguish between CP/M and AMSDOS files.

The addition of the disk system brings with it extra jumpblock entries, which are available to the user for disk management at the operating system level. These are detailed in the second half of the table.

Finally, we give here an example program that illustrates how information can be extracted from a file header. The file is first opened, using a call to CAS IN OPEN, which generates the header in RAM. Having extracted the relevant information, the file is then discarded using CAS IN ABANDON before any read takes place. The associated BASIC program serves to provide a user interface to the routine.

```
10 'File Header Reader
20 'Uses CAS IN OPEN at BC77H
30 routine=&8800: address=&8000: ON ERROR GOTO 530
40 MODE 1: MEMORY &5FFF
50 WHILE -1
60 CLS
70 LOCATE 10,10: PRINT "Tape or disc file? ";CHR$(18)
80 a$=""
90 WHILE a$<>"T" AND a$<>"D"
100 a$=UPPER$(INKEY$)
110 WEND
120 IF a$="T" THEN ITAPE ELSE IF a$="D" THEN IDISC
130 LOCATE 10,12: INPUT "Filename";file$
140 IF LEN(file$)=0 AND a$="D" THEN 130
150 GOSUB 420
160 CALL routine
170 buffer=PEEK(address)+256*PEEK(address+1)
180 LOCATE 20,12: ff=0
190 FOR name=0 TO 7-8*(t$="T")
200 char=PEEK(buffer+name)
210 IF (char<32 OR char>126) AND char>0  THEN  ff=1: GO
TO 230
220 PRINT CHR$(char);
230 NEXT
240 IF ff=1 THEN PRINT CHR$(8);"No filename";
250 PRINT
260 'test filetype
270 LOCATE 5,16: PRINT "Filetype: ": LOCATE 20,16
280 type=PEEK(buffer+18)
290 IF (type AND 1)=1 THEN PRINT "Protected ";
300 file = type AND &FE
310 IF file=0 THEN PRINT "Basic"
320 IF file=2 THEN PRINT "Binary"
330 IF file=6 THEN PRINT "ASCII"
340 IF file>6 THEN PRINT "UnKnown"
350 length=PEEK(buffer+24)+256*PEEK(buffer+25)
360 LOCATE 5,18: PRINT "Data length:":LOCATE 20,18: PRI
NT HEX$(length,4)
370 entry=PEEK(buffer+26)+256*PEEK(buffer+27)
380 LOCATE 5,20: PRINT "Entry address:": LOCATE 20,20:
PRINT HEX$(entry,4)
390 LOCATE 9,25:PRINT "Press a Key to continue"
400 WHILE INKEY$="":WEND
410 WEND
420 RESTORE 520: READ byte: off=0
430 WHILE byte<>999
440 POKE routine+off,byte
450 off=off+1:READ byte
460 WEND
470 POKE routine+1,LEN(file$)
480 FOR char=1 TO LEN(file$)
490 POKE routine+&100+char-1,ASC(MID$(file$,char,1))
500 NEXT
510 RETURN
520 DATA &06,&00,&21,&00,&89,&11,&00,&60,&cd,&77,&bc,&7
d,&32,&0,&80,&7c,&32,&01,&80,&cd,&7d,&bc,&c9,999
530 'error trap
540 IF ERL=80 THEN a$="":RESTORE
550 STOP
```

Since the early days of space invaders, when young boys first hunched over their VDUs blasting alien spaceships from the heavens, computer games have often been criticised for being excessively violent. Bobby Pickering looks at two packages for the Commodore 64 that sound as if they might appeal to pacifists

# NON-VIOLENT ACTION?

Given a rough outline of the game, you might be forgiven for thinking that Elidon (Orpheus, £8.95) has very little to attract a games player accustomed to fast action shoot-em-ups. Set in a beautiful and mysterious underground world consisting of nearly 300 different locations, your task is to guide the Faerie Queen through Elidon in search of the magic flowers needed to banish winter's cold shadow. It certainly sounds like the sort of scenario guaranteed to frighten off normally stout-hearted young men, but Elidon proves such a difficult and enchanting pursuit that even those toughened by years of playing war games have been known to fall under the fairy's magic spell.

The Faerie Queen begins the game with three sets of wings; she loses one of these each time her magic strength (indicated by a meter on the right of the screen) is exhausted. There are many perils for her in the Magic Kingdom — nasty moths, stinging nettles and creatures with blinking eyes in darkened dungeons — and she must be carefully steered clear of these. But she is most likely to be brought down to earth when you attempt to negotiate her through the room exits. If she hits her head, wings or wand, then she giddily plummets to the ground to recover her senses.

Throughout the fairy kingdom, with its varied and colourful fauna and flora, are scattered seven plants and seven bottles with magical potions in them. You must find the potions, pour them on the plants, and pluck the blooms that appear as a result of your ministrations. These must then be taken back to the garland ring that marked the start of your quest (each one safely deposited earns you another pair of wings). The game is available from Orpheus Ltd., The Smithy, Unit 1, Church Farm Estate, Hatley St George, Near Sandy, Beds SG19 3HP.

A game called Peace Women (Knightsoft, £7.95) sounds as if it might appeal to those suffering from shoot-em-up shell shock. But although the package bases itself on the real-life conflict between the peace women and the Ministry of Defence at Greenham Common, it really is a flight of fancy that is completely divorced from reality.

What the game's programmers want us to imagine (although the suspension of disbelief could be difficult) is that a peace woman has got hold of what looks like a Sinclair C5 Special and is motoring through the Berkshire countryside towards the peace camp. Determined to stop her, however, are endless battalions of MOD police on motor bikes, who approach her from ahead and behind. But the C5 is more than it seems; as the inlay points out, 'it is armed with ground to air missiles', and these can be used to pop off (literally) the pursuing bikers.

And so the action takes its tedious course until one of the motorbikes catches up to the C5. At this point a little routine takes over that generates a haloed feminine spirit ascending from the C5. In each game you have three 'lives', but after a while endlessly 'popping-off' vehicles has a limited engagement value.

Peace Women is marketed by Knightsoft, The Round Table, 87 Brunswick Street, Burnley, Lancashire BB11 3LE.

© 1983 AYDIN CONTROLS